An Introductory Lab in Embedded and Cyber-Physical Systems

Jeff C. Jensen
Edward Ashford Lee
Sanjit Arunkumar Seshia

Zeroth Edition, Version 0.05

Please cite this book as:

J.C. Jensen, E.A. Lee, and S.A. Seshia,
An Introductory Lab in Embedded and Cyber-Physical Systems,
http://leeseshia.org/lab, 2012.

Contents

Preface

What this Book is About

The theme of this book is the characterization of embedded and cyber-physical systems not by resource constraints, but rather by interactions with the physical world. While resource constraints are an important aspect of design, such constraints are part of every engineering discipline and give little insight into the interplay between computation and physical dynamics. Our approach draws from topics in physics, circuits, transducers, digital signal processing, digital communications, networking, operating systems, robotics, control theory, algorithms, probability, and logic. Through structured laboratory exercises, we aim to expose the lowest levels of abstraction for programming embedded systems such as traditional imperative programming models, through to the highest levels of abstraction such as graphical system design tools and concurrent models of computation.

The laboratory exercises in this text emphasize the basics of models, analysis tools, and design of embedded systems. Exercises guide students in modeling the physical world with continuous-time differential equations and modeling computations using logic and discrete models such as state machines. These modeling techniques are evaluated through the use of meta-modeling, illuminating the interplay of practical design with formal models of systems that incorporate both physical dynamics and computation. Formal techniques are introduced to specify and verify desired behavior. A combination of structured labs and design projects solidifies these concepts when applied to the design of embedded and cyber-physical systems with real-time and concurrent behaviors.

The theoretical foundation for this text follows Lee and Seshia, *Introduction to Embedded Systems: A Cyber-Physical Approach* (Lee and Seshia, 2011).

What is Missing

This is a work in progress, and the actual impact of laboratory exercises is difficult to measure objectively. Students we have taught are generally interested and engaged, celebrating their projects as well as those from other teams. They are often proud of what they accomplish, and even post project presentation videos to the internet. Such anecdotes give some insight into the impact of a course, but how do we know for sure whether a particular change in the course or laboratory design is actually an improvement? We are pleased, at least, to witness through the course that students surprise both themselves and their instructors, that projects demonstrate an understanding of the theoretical concepts introduced in lecture, and that students have received job offers from industry mentors.

A kit-based approach to the lab that is commercially supported and costs about the equivalent of a traditional textbook could go a long way towards facilitating export of laboratory curriculum. The iRobot Create meets these requirements, and greatly influenced laboratory development. We interfaced the internal microprocessor on the iRobot Create with a more advanced external controller, for better or worse, only because it was underpowered and difficult to reprogram, though we have no reason to believe that students benefit educationally from a more powerful processor. We seek a kit that offers mobility, sensing, and actuation together with a suitably powerful and adaptable embedded controller, as a platform for model-based design of cyber-physical systems.

How to Use this Book

The electronic version of this book includes a number of hyperlinks to documents that are needed or helpful for completing the labs. These documents are provided as downloads at `http://LeeSeshia.org/lab`. They are also cited in the references section at the end of the book, where URLs to the original document are provided.

Laboratory exercises in this text are designed to be modular so that they may be chosen according to the topics that best suit a course. Dependencies between labs are explicitly stated, and generally indicate two or more labs fit into a sequence. Software and hardware tools used are listed at the beginning of each laboratory, and are covered in Appendix A: Lab Software and Appendix B: Lab Hardware. Setup of laboratory workstations and equipment is covered in Appendix C: Lab Setup.

Each structured laboratory consists of a pre-laboratory assignment and in-laboratory exercises. The pre-laboratory exercises are *critical* preparation in advance of in-laboratory exercises, as they introduce documentation, tools, and concepts used. We suggest formal laboratory sessions begin with a brief lecture that covers the instruments used, their theory of operation, and the overall goal of the laboratory. Teams of two or three may then begin working on assigned laboratory exercises; in many cases, solutions are not unique, and we encourage teams to experiment and innovate.

Intended Audience

The laboratory exercises in this book were designed and piloted for the course EECS 149, "Introduction to Embedded Systems" at the University of California, Berkeley (Lee and Seshia, 2010; Jensen et al., 2011; Lee et al., 2012). The course is targeted at advanced undergraduate juniors and seniors in Electrical Engineering and Computer Science. Prerequisites include an introductory course in signals and systems, an introductory course in computer architecture (which covers both C and assembly programming), and an introductory course in discrete mathematics. While these prerequisites establish a common language for the technical aspects of embedded systems, we believe that the ubiquitous and interdisciplinary nature of embedded systems requires students to investigate topics beyond computer science.

Acknowledgements

Many people have contributed to the design and debugging of the exercises in this book. Trung N. Tran and Godpheray Pan from National Instruments provided critical help configuring and adapting the versatile Single-Board RIO and the software infrastructure, and customizing a physics simulator and renderer for a complete model-based design workflow. Additional key contributions were made by the Electronics Support Group (ESG)

in the EECS Department at UC Berkeley, most particularly Ferenc Kovac and Winthrop Williams. Support with software and networking infrastructure was provided by Linda Huang and Kevin Mullally. We also thank the graduate student instructors (GSIs) who supervised initial offerings of the course and help to develop and debug these lab exercises, particularly Dai Bui, Shanna-Shaye Forbes, Ben Lickly, Wenchao Li, Isaac Liu, Kevin Weekly, and Michael Zimmer.

1

Sensor Interfacing and Calibration

Contents

The exercises in this chapter guide you in interfacing and calibrating sensors. These introductory exercises focus on key concepts, particularly on identifying bias and sensitivity, explained in Lee and Seshia (2011), Draft Chapter on Sensors and Actuators.

1.1 Interface to and Calibrate the WiiMote

This lab focuses on accelerometers. We will use the Nintendo **Wii remote**, known informally as a **WiiMote**, which includes a three-axis accelerometer. You will interface your computer with the WiiMote via a **Bluetooth** wireless link and obtain the raw data from the accelerometer. The goal of the lab is to learn how to interpret this raw data by calibrating the sensor. As a side benefit, you will begin to get some exposure into the process of connecting a computer to external devices to read sensor data and control actuators.

1.1.1 Suggested Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

1. Introduction to Embedded Systems (Lee and Seshia, 2011)
   (a) Chapter 9: Input and Output, §9.3 (The Analog/Digital Interface)
   (b) Chapter 6: Concurrent Models of Computation, §6.3.4 (Structured Dataflow)
   (c) Draft Chapter on Sensors and Actuators.
      i. §7.1 (Models of Sensors and Actuators)
      ii. §7.2.2 (Measuring Tilt and Acceleration)
2. Implementing a Tilt-Compensated eCompass (Ozyagcilar, 2011)
   (a) §3 (Accelerometer and Magnetometer Outputs)
   (b) §4 (Tilt-Compensation Algorithm)
   (c) §7.2 (Modulo Arithmetic Low Pass Filter)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. We refer you to key topics, but you may need to reference other sections to complete these exercises.

1. Appendix A: Lab Software, §A.1 (LabVIEW)
2. Appendix A: Lab Software, §A.1.2 (LabVIEW MathScript)
3. Appendix B: Lab Hardware, §B.4 (Nintendo Wii Remote)
4. Developing Algorithms Using LabVIEW MathScript RT Module (National Instruments, 2010a)
5. Wii Remote on Wikipedia (Wikipedia, 2012c)
6. WiiBrew WiiMote wiki (WiiBrew, 2011)
7. Blueooth HID Profile (Bluetooth Special Interest Group, 2003)

(a) "Introduction to the HID Protocol", p. 18-23
(b) "BT HID Transaction Header", p. 57-64

1.1.2 Equipment and Software Required

1. PC computer running Microsoft Windows XP, Vista, or 7.
2. Nintendo Wii Remote (B.4)
3. Bluetooth radio on the PC computer (either built in or external).
4. National Instruments LabVIEW 2011 or later. (A.1)

1.1.3 Prelab Exercises

1. Introduction to Embedded Systems (Lee and Seshia, 2011)
   (a) Draft Chapter on Sensors and Actuators (all exercises)
2. Appendix B: Lab Hardware, §B.4.2 (Exercises) (all)

1.1.4 Lab Exercises

The following exercises reference files that may be downloaded from the Jensen, Lee, and Seshia website,

`http://LeeSeshia.org/lab/releases/0.05`

Be sure to completely extract all archives before opening any files, or you may receive dependency errors. Once extracted, the exercises may be found in the folder

`LabVIEW/wiiMote`

1. *Pair a desktop computer with the WiiMote:* **WiiMote Pair.vi** pairs your desktop computer with your WiiMote, enabling communication. Given the MAC address of your WiiMote, the VI searches available Bluetooth devices and will pair your device if its MAC address is found. Though the Windows operating system provides an interface to pair Bluetooth devices, using this VI to pair by MAC address eliminates confusion that arises when multiple WiiMotes are available.

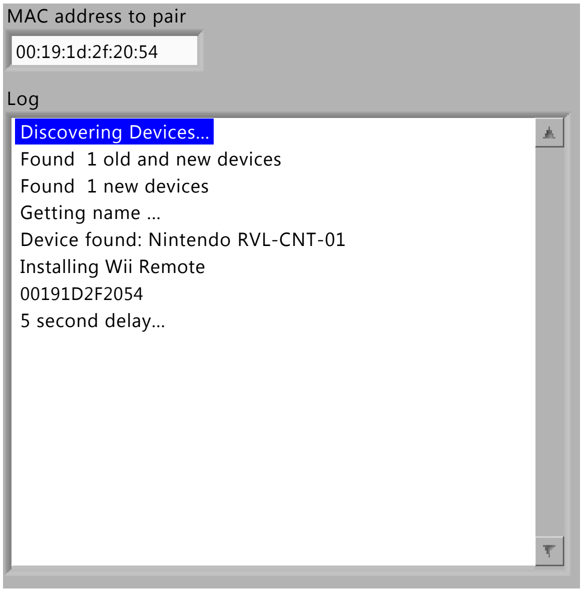
Open WiiMote Pair.vi. You do not need to modify or view the block diagram for this exercise. Key in the MAC address of your WiiMote, and run the VI to pair. When pairing the WiiMote, you must repeatedly and simultaneously press the 1 and 2 user buttons for the entire pairing process. Continue to press and let go until the after VI has completed (Fig. 1.1). When correctly paired, you should see a Windows notification that a Bluetooth HID driver was installed, and the WiiMote should flash all four user LEDs continuously. If the LEDs do not flash continuously, run the pair VI again.

(a) What is the MAC address of your WiiMote?

2. *Plot the uncalibrated accelerometer signal from the WiiMote:* after pairing, open and run **WiiMote Interface.vi** (Fig. 1.2). You do not need to modify this VI for this exercise.

(a) What are the values of the x and y axes when the WiiMote is at rest on level ground?

(b) What is the value of the z axis when the WiiMote is at rest on level ground? What quantity is being measured?



```
00:19:1d:2f:20:54
```

```
Discovering Devices...
Found  1 old and new devices
Found  1 new devices
Getting name ...
Device found: Nintendo RVL-CNT-01
Installing Wii Remote
00191D2F2054
5 second delay...
```

Figure 1.1: WiiMote Pair.vi, showing successful pairing with the WiiMote.

3. *Control the WiiMote actuators:* In the Front Panel of WiiMote Interface.vi, modify the values for Report ID and Payload to toggle user LEDs, and to toggle the rumble motor.

Note: After an instruction has been sent to the WiiMote to turn off all four user LEDs, the device appears to be powered off even though it is still running. To preserve battery life, hold down the power button for three seconds when the device is no longer in use.

(a) What is the Report ID and Payload to turn WiiMote LEDs 2 and 4 on, and LEDs 1 and 3 off?

(b) How can you modify the above sequence to also enable the rumble motor?

4. *Measure sensitivity and bias of the accelerometer:* The buttons on the front panel are associated with panes in the Event Structure. To view the subdiagrams for each event handled by the event structure, use the scroll arrows next to the name of the event handled. Modify the subdiagrams of the Event Structure so that, when the 'measure bias' or 'measure sensitivity' button is pressed on the front panel, the corresponding measurement recorded and displayed in the 'calibration (raw)' control. You may assume the sensitivity

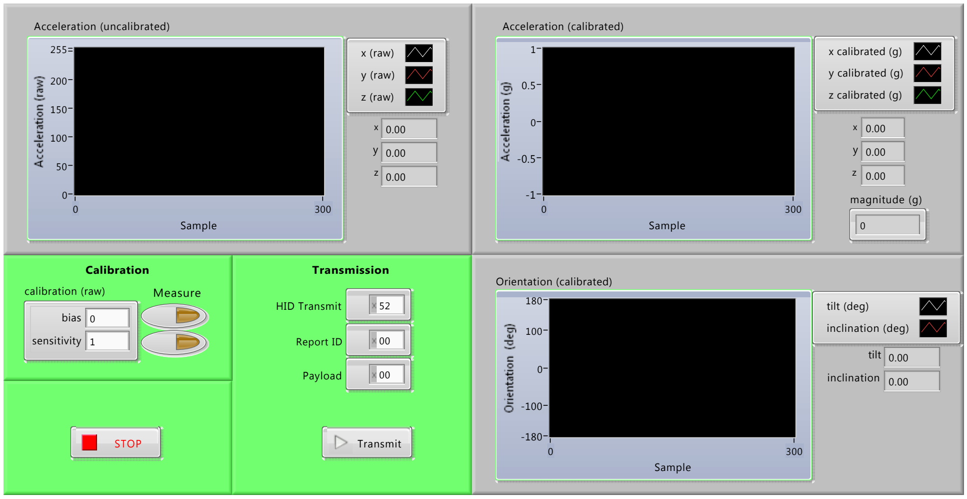


Figure 1.2: WiiMote Interface.vi Front Panel.

and bias are approximately the same for all three axes. You do not need to create new buttons or events for this exercise.

Hint: To retain your calibration parameters after LabVIEW is closed, right click on the "calibration (raw)" control on the front panel and select "Data Operations - Make Current Values Default".

(a) Where are these measurements stored?

(b) What bias did you record?

(c) What sensitivity did you record?

(d) Provide screenshots of changes to the block diagram code.

5. *Calibrate the accelerometer:* modify MathScript Node in WiiMote Interface.vi so that the "Accelerometer (calibrated)" chart plots acceleration of the x, y, and z axes, and the "magnitude" indicator displays the magnitude of acceleration, all in units of g.

(a) What equations did you use to calibrate the accelerometer and to calculate the magnitude of the acceleration it measures?

(b) Based on your calibration parameters and the number of bits of resolution of the WiiMote ADC, and an affine model of the WiiMote accelerometer, what is your estimate of the maximum acceleration that the WiiMote can measure?

(c) When the WiiMote is at rest, what should be the value of the "magnitude" indicator?

(d) Why is it better to perform calibration at runtime, rather than hard-coding values for the sensitivity and bias of the sensor?

(e) Provide the relevant MathScript code and screenshots of changes (if any) to the block diagram.

6. *Measure Pitch and Roll:* modify the MathScript Node in WiiMote Interface.vi so that the "Orientation (calibrated)" chart plots pitch and roll of the WiiMote, in units of degrees.

Hint: The coordinate system used in Implementing a Tilt-Compensated eCompass (Ozyagcilar, 2011) and the coordinate system used by the WiiMote (Fig. B.11) are related by swapping the x and y axes.

(a) What equations did you use to calculate pitch and roll?

(b) Provide the relevant MathScript code and screenshots of changes (if any) to the block diagram.

7. *Filter the accelerometer signal:* Accelerometers are often used to detect high frequency signals such as noise and vibration; for a game controller, lower-frequency movements of the user should be isolated. In particular, the rumble motor will introduce a high-frequency signal. Implement a lowpass filter to smooth the signal of the calibrated accelerometer when the rumble motor is on.

Hint: A simple lowpass filter is discussed in Implementing a Tilt-Compensated eCompass (Ozyagcilar, 2011).

(a) Describe your filter, and include an equation for its impulse response, frequency response, or its output as a function of input.

(b) Provide the relevant MathScript code and screenshots of changes (if any) to the block diagram.

8. *Share your Feedback:* what did you like about this lab, and what would you change?

1.1.5 Troubleshooting

1. **I receive Error 1172, "WiimoteLib.WiimoteNotFoundException" when running WiiMote Interface.vi.** This error occurs if your desktop is not paired with a WiiMote. Run WiiMote Pair.vi to pair again.

2. **I am able to pair the WiiMote, but WiiMote Interface.vi either does not show input or returns Error 1172 "WiimoteLib.WiimoteNotFoundException".** Check the WiiMote batteries – when low, the device may stay powered on long enough to pair, only to power off shortly thereafter. When first powered on and paired, the four LEDs should remain flashing until the device is powered off, or the LED state is explicitly changed by a Bluetooth command.

3. **I am able to pair the WiiMote and run WiiMote Interface.vi, but after a few seconds the output does not change in response to movement of the controller.** Check the WiiMote batteries – when low, the device may stay powered on long enough to pair, only to power off shortly thereafter. When first powered on and paired, the four LEDs should remain flashing until the device is powered off, or the LED state is explicitly changed by a Bluetooth command.

4. **I receive Error 1172, "Timed out waiting for status report".** This may occur if you are using a WiiMote clone (i.e. not manufactured by Nintendo), and are not using the WiiMoteLib binary distributed with the lab materials. In initializing the WiiMote, the open-source WiiMoteLib verifies a vendor ID, which differs between authentic and cloned WiiMotes. We modified the WiiMoteLib source to omit this call during initialization, and provide the source as part of the downloadable lab materials.

5. **I receive Error 1172, "Request for the permission of type... failed".** This is a Microsoft .NET exception, indicating WiimoteLib.dll could not be loaded from a network drive. If possible, try moving your lab files to a local (i.e. `C:`) drive. If your administrator has placed WiimoteLib.dll in a system directory, it is possible that the WiimoteLib.dll is loading from your lab directory first; try renaming the file `WiimoteLib.dll` to `WiimoteLib.dll.bak` and relaunch LabVIEW.

6. **My VI is broken and raises the error, "Invalid Procedure" or "Function Not Found" on the WriteStatusReport node.** This may occur if you are not using the WiiMoteLib binary distributed with the lab materials. The standard library does not allow external programs to transmit arbitrary status report packets. We

modified the WiiMoteLib source to add this functionality, and provide the source as part of the downloadable lab materials. Make sure there are no other instances of the WiiMoteLib library in the system path.

2 Programming of Embedded Systems

Contents

The exercises in this chapter guide you in programming embedded systems.

2.1 Interface to MicroBlaze

This lab focuses on interfacing with an embedded microcontroller in C. Your software executes “bare-iron”, or in the absence of a scheduling kernel or operating system, and has exclusive access to the microcontroller processor, memory, and peripherals. The goal is to configure processor interrupts to periodically sample a sensor, a fundamental process in any cyber-physical system.

Interrupts can be used to periodically sample sensors such as microphones for voice control or ultrasonic range finders for navigation. They can be used to drive actuators by generating a pulse-width modulation (**PWM**) signal for sound or motor control, or to simply flash an LED to signal a device fault. Software written for an embedded system may even reside almost entirely in timed interrupts.

These exercises will guide you in configuring timed interrupts via memory-mapped registers on an embedded microcontroller. For many, the first experience programming interrupts is challenging: interrupts are documented in verbose documentation, the embedded device being programmed may not have a display or debugging interface, and breaking from the sequential model of traditional programming languages introduces new complexities (Lee, 2006). Inherent in these exercises is the importance of navigating technical documentation to solve a problem.

The exercises are broken into two parts: first, you will configure a timed interrupt to poll an ADC. Next, you will configure a timed interrupt to trigger an ADC conversion and a second interrupt to fire when the ADC conversion is complete. We suggest connecting an analog sensor to the ADC so real-world values are read.

2.1.1 Suggested Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

1. Introduction to Embedded Systems (Lee and Seshia, 2011)
   (a) Chapter 7: Embedded Processors, §7.1.1 (Microcontrollers)
   (b) Chapter 8: Memory Architectures, §8.2.1 (Memory Maps)
   (c) Chapter 8: Memory Architectures, §8.2.2 (Register Files)
   (d) Chapter 9: Input and Output, §9.2 (Sequential Software in a Concurrent World)
2. MicroBlaze Interrupts and the EDK (Hickok, 2009)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. We refer you to key topics, but you may need to reference other sections to complete these exercises.

1. Appendix B: Lab Hardware, §B.3 (National Instruments Single-Board RIO 9606)
2. Appendix B: Lab Hardware, §B.3.2 (Analog RMC)
3. Appendix B: Lab Hardware, §B.5 (Xilinx MicroBlaze)
4. Embedded System Tools Reference Guide (Xilinx, 2010a)
   (a) Chapter 9: GNU Compiler Tools, §"Interrupt Handlers", p.133.
5. LogiCORE IP XPS Interrupt Controller (Xilinx, 2010b)
   (a) §"Interrupt Controller Core", p.4
   (b) §"Programming Model", p.9-15
6. LogiCORE IP XPS Timer/Counter (Xilinx, 2010c)
   (a) §"Functional Description", p.2-5
   (b) §"Register Descriptions", p.12-15

2.1.2 Equipment and Software Required

1. PC computer running Microsoft Windows XP, Windows Vista, or Windows 7
2. Xilinx SDK 12.4
3. National Instruments Single-Board RIO
   (a) sbRIO-9606 embedded microcontroller (B.3)
   (b) PS-2, PS-3, or PS-15 power supply
   (c) Analog RMC (C.3.2)
4. Xilinx Platform Cable USB II (JTAG)

2.1.3 Prelab Exercises

1. Introduction to Embedded Systems (Lee and Seshia, 2011)
   (a) Chapter 9: Input and Output, Ex. #1-3
2. Appendix B: Lab Hardware, §B.3 (National Instruments Single-Board RIO 9606) (all exercises)
3. One way to read from an ADC is to use a loop in the `main()` function to trigger an ADC conversion, wait for completion, read its result, and print the result to the screen.

   (a) Is this concurrent or sequential software?

   (b) Are processor resources well-utilized by this method?

   (c) Is timing explicit in this method? In what ways could you attempt to achieve a fixed sampling rate?

4. A second way to read from an ADC is to use timed interrupts. A periodic timer ISR triggers an ADC conversion, waits for completion, and stores its result in a global variable. The `main()` program loop is tasked with printing the result to the screen.

   (a) Is this concurrent or sequential software?

   (b) Are processor resources well-utilized by this method?

   (c) Is timing explicit in this method? In what ways could you attempt to achieve a fixed sampling rate?

5. A third way to read from an ADC is to use timed interrupts and ADC interrupts. A periodic timer ISR triggers an ADC conversion and returns. When the ADC conversion is complete, an ADC interrupt ISR reads the result and returns. The `main()` program loop is tasked with printing the result to the screen.

   (a) Is this concurrent or sequential software?

   (b) Are processor resources well-utilized by this method?

   (c) Is timing explicit in this method? In what ways could you attempt to achieve a fixed sampling rate?

6. What are some of the major differences between using polling and using interrupts to read from an ADC?
7. What is the meaning of the `interrupt_handler` compiler attribute?

2.1.4 Lab Setup

The following exercises reference files that may be downloaded from the Jensen, Lee, and Seshia website,

`http://LeeSeshia.org/lab/releases/0.05`

Be sure to completely extract all archives before opening any files, or you may receive dependency errors. Once extracted, the exercises may be found in the folder

`C:/users/ee149-XX/microblazeInterface0_05.zip`

Note: Xilinx SDK 12.4 seems to be unable to compile from a network location. When prompted to create or open a workspace, be sure to point to the `C:\users` drive. *Be sure to backup your solutions to your user drive U: .*

1. *Verify the Hardware Setup:* Your sbRIO should be powered, and your Xilinx JTAG device should be connected to your desktop computer via USB. The other end of the JTAG device should match the configuration in Table B.1.

2. *Configure a Workspace:* Open Xilinx SDK. The SDK will prompt you to select a workspace, as shown in Fig. 2.1.

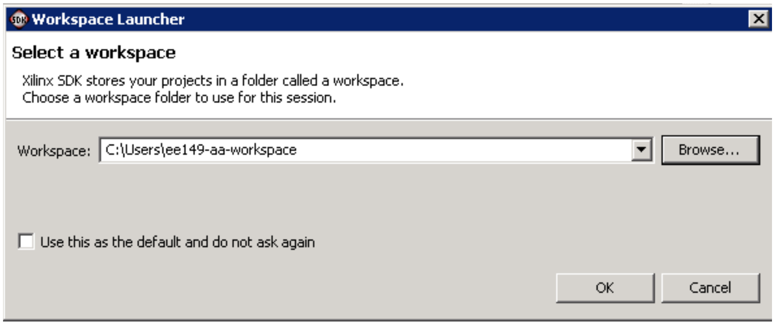
Select a folder on a local drive to use as your workspace. The path *must not contain any spaces*. This is a limitation of the Xilinx software, and selecting a folder whose path contains spaces will result in problems compiling.

After creating (or selecting) a workspace, the Xilinx SDK environment will open. From the top menu bar, select "Window, Show View, Project Explorer" to change to a default window configuration (Fig. 2.2).

You will now import project templates into you workspace. Right click in an empty part of the Project Explorer and select "Import..." to launch the Import Wizard. The type of project you will import is "General, Existing Project into Workspace" (Fig. 2.4).

Select "Next" to browse for the project to import. Choose "Select archive file", press the "Browse" button, and navigate to

`microblazeInterface0_05.zip`



Workspace Launcher

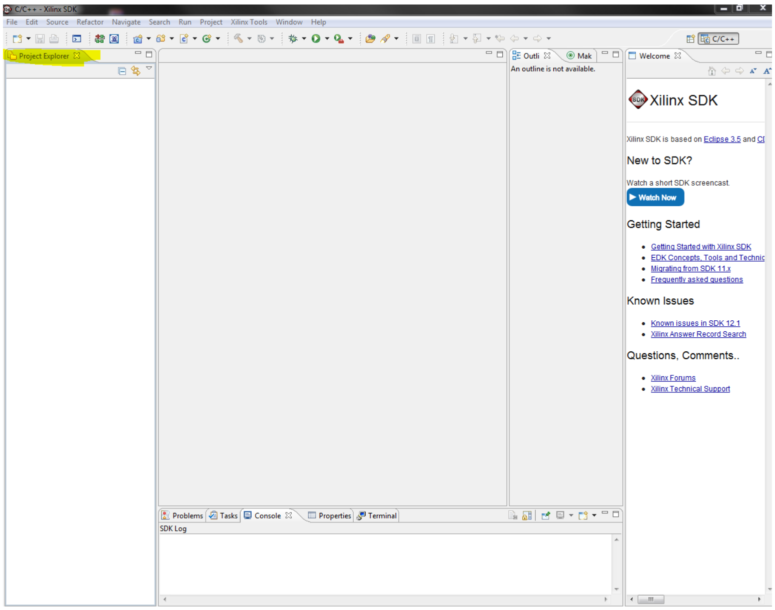
Select a workspace

Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\ee149-aa-workspace

Use this as the default and do not ask again

Figure 2.1: Xilinx SDK Workspace Launcher.



An outline is not available.

SDK Log

Xilinx SDK

Xilinx SDK is based on Eclipse 3.5 and CI

New to SDK?

Watch a short SDK screencast.

Getting Started

- Getting Started with Xilinx SDK
- EDK Concepts, Tools and Technic
- Migrating from SDK 11.x
- Frequently asked questions

Known Issues

- Known issues in SDK 12.1
- Xilinx Answer Record Search

Questions, Comments..

- Xilinx Forums
- Xilinx Technical Support

Figure 2.2: Xilinx SDK Project Explorer view.

The Projects list will populate; ensure all projects are selected and click “Finish”. The project will now appear in the Project Explorer view, and will automatically build.

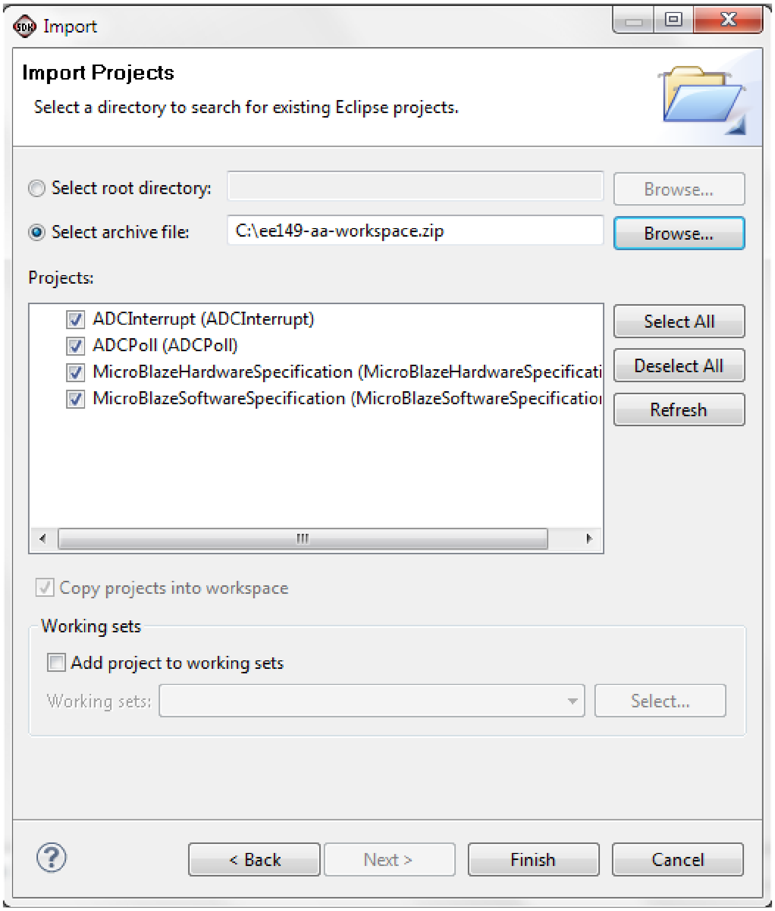
You will see a project titled `MicroBlazeHardwareSpecification` and another titled `MicroBlazeSoftwareSpecification`. `MicroBlazeHardwareSpecification` is a hardware description project that contains information about sbRIO and how MicroBlaze should be implemented on it. `MicroBlazeSoftwareSpecification` is a board support package, which provides drivers to hardware peripherals on sbRIO. You do not need to modify either of these projects, but feel free to explore their content.

3. *Run:* Select the polling project in the Project Explorer, and from the top menu bar select “Project, Build All”. Then, right click on the project in the Project Explorer and select “Run, Run Configurations...”. Right click on Xilinx C/C++ ELF and select ’New’. Under the STDIO Connection tab, check the box “Connect STDIO to Console” and change port to “JTAG UART”.

You may receive the warning “FPGA not configure with a bitstream in this SDK session”. This warning indicates that the Xilinx SDK was not used to program the FPGA. This warning may be dismissed, since the fixed personality containing MicroBlaze is properly configured to interact with the Xilinx SDK. Check the box “FPGA configured outside SDK. Do not show this message again.” and click ”Yes” to continue.

You may also receive a warning that you do not have administrator privileges when using the XMD debug console with the SDK. You may disregard this warning.

If the program compiles and downloads successfully, you should see console output appear at the bottom of the Project Explorer view. The template code will compile and download to the target without modification, but its functionality is incomplete.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

( ) Select root directory:

[✓ Select archive file:] C:\ee149-aa-workspace.zip

Projects:

- ☑ ADCInterrupt (ADCInterrupt)
- ☑ ADCPoll (ADCPoll)
- ☑ MicroBlazeHardwareSpecification (MicroBlazeHardwareSpecificati
- ☑ MicroBlazeSoftwareSpecification (MicroBlazeSoftwareSpecificatio

☑ Copy projects into workspace

Working sets

☐ Add project to working sets

Working sets:

Figure 2.3: Xilinx SDK Import Wizard, browsing for a project archive.

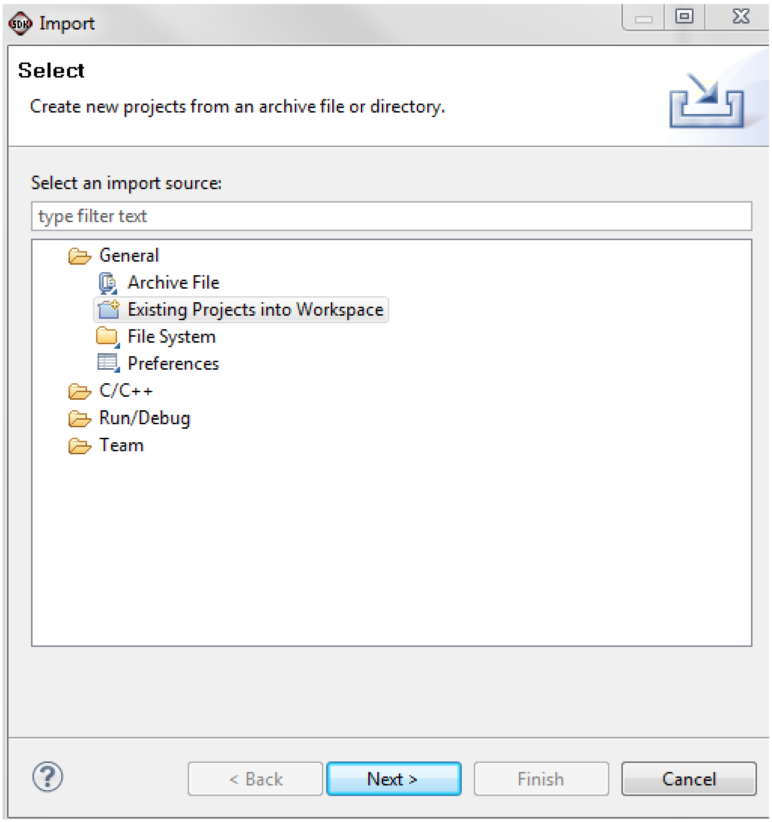
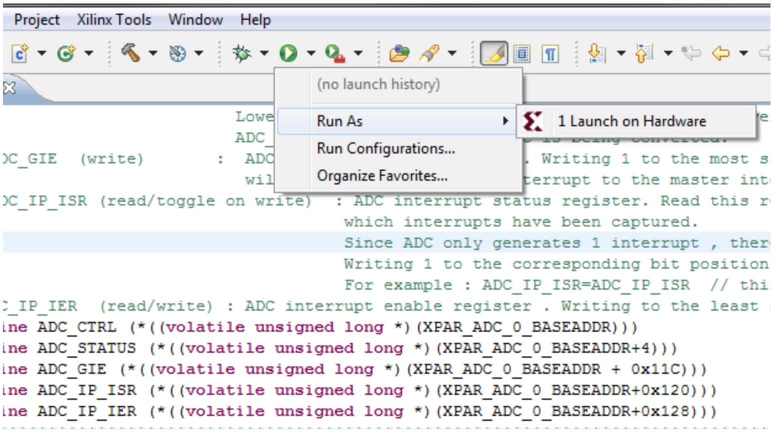
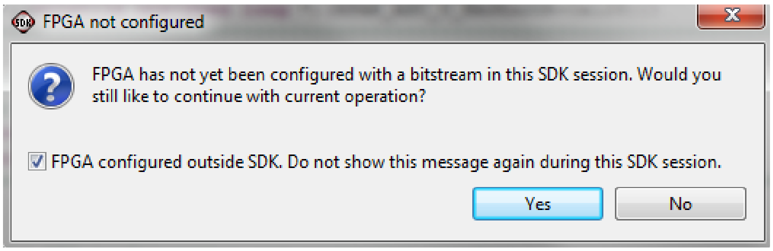


Figure 2.4: Xilinx SDK Import Wizard, set to import an existing project into a workspace.



```
                      Lowe
                      ADC_
)C_GIE   (write)         :  ADC                        . Writing 1 to the most s
                            wil                        terrupt to the master int
)C_IP_ISR (read/toggle on write)  : ADC interrupt status register. Read this r
                                    which interrupts have been captured.
                                    Since ADC only generates 1 interrupt , ther
                                    Writing 1 to the corresponding bit position
                                    For example : ADC_IP_ISR=ADC_IP_ISR  // thi
:_IP_IER  (read/write) : ADC interrupt enable register . Writing to the least
ine ADC_CTRL (*((volatile unsigned long *)(XPAR_ADC_0_BASEADDR)))
ine ADC_STATUS (*((volatile unsigned long *)(XPAR_ADC_0_BASEADDR+4)))
ine ADC_GIE (*((volatile unsigned long *)(XPAR_ADC_0_BASEADDR + 0x11C)))
ine ADC_IP_ISR (*((volatile unsigned long *)(XPAR_ADC_0_BASEADDR+0x120)))
ine ADC_IP_IER (*((volatile unsigned long *)(XPAR_ADC_0_BASEADDR+0x128)))
```

Figure 2.5: Download and run .


FPGA not configured

FPGA has not yet been configured with a bitstream in this SDK session. Would you still like to continue with current operation?

☑ FPGA configured outside SDK. Do not show this message again during this SDK session.

Figure 2.6: A warning may occur when using the MicroBlaze fixed personality, and may be dismissed.

2.1.5 Lab Exercises

An interrupt service routine (ISR) should contain a minimum number of instructions. An ISR that is invoked periodically and has an execution time nearly as long as its period will starve other processes; worse, an ISR whose execution time is *equal to or greater* than its period may cause unpredictable or unstable behavior. Interrupts step outside the paradigm of sequential code, and care must be taken to ensure no two processes attempt to simultaneously access shared resources. It is important to document any shared resources used by an ISR, including whether or not the ISR will read or write to a resource.

It is difficult to debug interrupts since their behavior may be intimately tied to the timing of the system and the occurrence of external events. A common mistake is to use a costly debugging technique like printing to a console or communication port in the body of an ISR; such operations are processor-intensive and access shared hardware resources, with the potential of contributing significant timing and logical artifacts. Any debugging instructions in an ISR will affect timing, but there are reasonably safe debugging mechanisms that will have minimal impact such as toggling a digital output line or incrementing a counter, both of which may be executed in a single processor cycle. The framework provided for this lab uses such techniques.

Your sbRIO is connected to an analog accelerometer, whose x, y, and z axes are connected to ADC channels 0, 1, and 2, respectively.

1. *Poll the ADC:* From the `ADCPoll` project, open the file `adcpoll.c`. This is the only file you need to modify to complete this exercise.

Configure the `main()` program loop to continuously poll the ADC and display the results to the debug console. You should not configure any interrupts for this exercise.

(a) Provide the content your `main()` program loop.

(b) When configuring the ADC and `main()` program loop, were there any steps that set the rate at which the ADC is polled, or does your code run as fast as possible?

2. *Use Timed Interrupts to Poll the ADC:* From the `ADCPollwithTimer` project, open the file `adcpolltimer.c`. This is the only file you need to modify to complete this exercise. You may reuse code from previous exercises.

Configure a timed interrupt to poll the ADC every 5 milliseconds. You should not poll the ADC in the `main()` program loop.

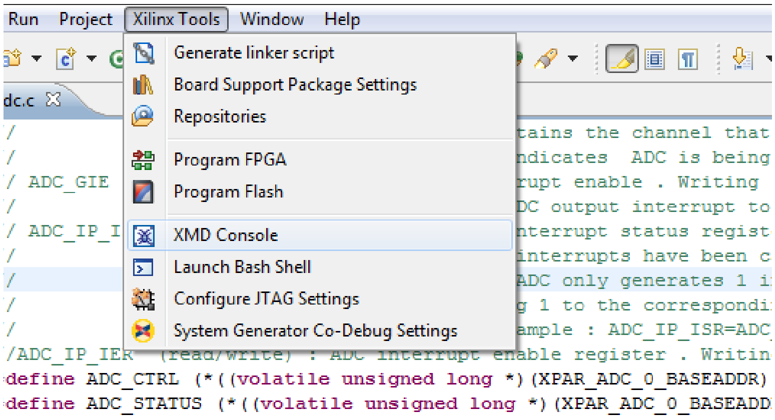
(a) Provide the content your `main()` function needed to configure the timer ISR, as well as the content of the timer ISR.

3. *Use ADC and Timed Interrupts to Read the ADC:* From the `ADCPollInterrupt` project, open the file `adcinterrupt.c`. This is the only file you need to modify to complete this exercise. You may reuse code from previous exercises.

Configure a timed interrupt to trigger an ADC conversion every 5 milliseconds, but do not poll in the timer ISR; instead, configure the ADC to generate an interrupt when a conversion is complete, and configure the ADC ISR to read the value. You should not poll the ADC in the `main()` program loop.

(a) Provide the content your `main()` function needed to configure the timer and ADC ISRs, as well as the content of the timer and ADC ISRs.

4. *Share your Feedback:* what did you like about this lab, and what would you change?



```
/                                    tains the channel that
/                                    ndicates  ADC is being
/ ADC_GIE                            rupt enable . Writing
/                                    DC output interrupt to
/ ADC_IP_I                           nterrupt status regist
/                                    interrupts have been c
/                                    ADC only generates 1 i
/                                    g 1 to the correspondi
/                                    ample : ADC_IP_ISR=ADC
/ADC_IP_IER  (read/write) : ADC interrupt enable register . Writin
define ADC_CTRL (*((volatile unsigned long *)(XPAR_ADC_0_BASEADDR)
define ADC_STATUS (*((volatile unsigned long *)(XPAR_ADC_0_BASEADD
```

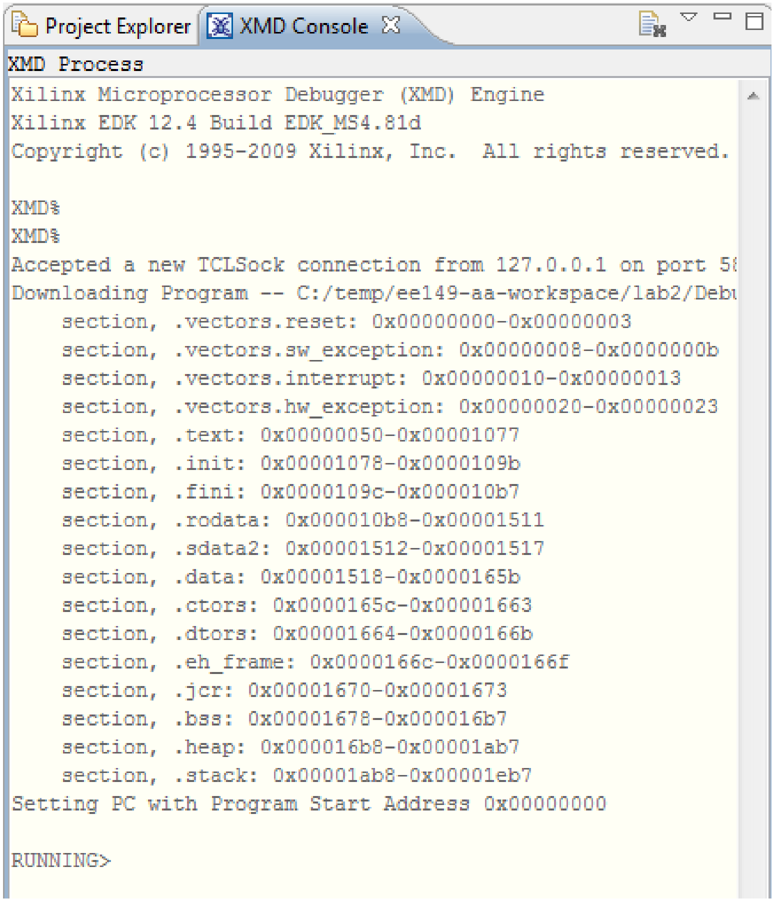
Figure 2.7: Opening the debug console.

2.1.6 Troubleshooting

1. **I receive the error “Failed to Open JTAG Cable” when attempting to run on target:** Check that the JTAG cable is connected to your computer, and double-check the connection to the target shown in Table B.1.

2. **I receive the error, “ERROR: Cannot perform the Debug Command, Current Processor State is ’Running’ ”.** This is a Xilinx SDK bug, and is resolved by restarting the SDK and rebooting the sbRIO.

3. **I cannot see the debug console:** `print()` statements in software are relayed via JTAG to your desktop computer. The Xilinx XMD Console displays these messages. First, be sure that the Project Explorer view is active. If the console is still not visible, it can be launched manually. From the top menu bar, select “Xilinx Tools, XMD Console”. The console is then connected to your target by entering the text `terminal` at the `XMD:` prompt and pressing enter (Fig. 2.7–2.8). A console window should appear, and will display any `print()` messages generated by software running on MicroBlaze.



```
Xilinx Microprocessor Debugger (XMD) Engine
Xilinx EDK 12.4 Build EDK_MS4.81d
Copyright (c) 1995-2009 Xilinx, Inc.  All rights reserved.

XMD%
XMD%
Accepted a new TCLSock connection from 127.0.0.1 on port 5
Downloading Program -- C:/temp/ee149-aa-workspace/lab2/Deb
    section, .vectors.reset: 0x00000000-0x00000003
    section, .vectors.sw_exception: 0x00000008-0x0000000b
    section, .vectors.interrupt: 0x00000010-0x00000013
    section, .vectors.hw_exception: 0x00000020-0x00000023
    section, .text: 0x00000050-0x00001077
    section, .init: 0x00001078-0x0000109b
    section, .fini: 0x0000109c-0x000010b7
    section, .rodata: 0x000010b8-0x00001511
    section, .sdata2: 0x00001512-0x00001517
    section, .data: 0x00001518-0x0000165b
    section, .ctors: 0x0000165c-0x00001663
    section, .dtors: 0x00001664-0x0000166b
    section, .eh_frame: 0x0000166c-0x0000166f
    section, .jcr: 0x00001670-0x00001673
    section, .bss: 0x00001678-0x000016b7
    section, .heap: 0x000016b8-0x00001ab7
    section, .stack: 0x00001ab8-0x00001eb7
Setting PC with Program Start Address 0x00000000

RUNNING>
```

Figure 2.8: Connected debug console.

3

Design of Cyber-Physical Systems

Contents

3.1 iRobot Navigation in C

This lab focuses on programming of cyber-physical systems. You will program an external embedded controller to control and interface with the iRobot Create robotics platform. The goal of this lab is to implement a state machine that instructs the robot to maintain a basic sense of orientation while avoiding obstacles.

3.1.1 Suggested Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

1. Introduction to Embedded Systems (Lee and Seshia, 2011)
   (a) Chapter 3: Discrete Dynamics, §3.3 (Finite-State Machines)
   (b) Chapter 7: Embedded Processors, §7.1.1 (Microcontrollers)
   (c) Chapter 7: Embedded Processors, "Circular Buffers" sidebar p.185
   (d) Chapter 9: Input and Output, §9.1.3 (Serial Interfaces)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. We refer you to key topics, but you may need to reference other sections to complete these exercises.

1. Appendix B: Lab Hardware, §B.2 (iRobot Create)
2. iRobot Create Owner's Guide (iRobot, 2006b)
3. iRobot Create Open Interface (OI) Specification (iRobot, 2006a)
4. Appendix B: Lab Hardware, §B.3 (National Instruments Single-Board RIO 9606)
5. Wind River Workbench User's Guide (Wind River Systems, 2006)
   (a) Chapter 1: Introduction §1.4 (Understanding Cross-Developing Concepts)
   (b) Chapter 19: Connecting to Targets §19.5 (Establishing a Connection)
   (c) Chapter 19: Connecting to Targets §19.6 (Connect to the Target)
   (d) Chapter 23: Launching Programs §23.2.8 (Using Your Launch Configuraiton)
   (e) Chapter 25: Debugging Projects
   (f) Chapter 26: Troubleshooting §26.4.3 (Target Manager Errors)

3.1.2 Equipment and Software Required

1. PC computer running Microsoft Windows XP

2. Wind River Workbench 2.5 for VxWorks (Wind River Systems, 2012)
3. Cal Climber (C.3.4)
    (a) iRobot Create (B.2)
    (b) National Instruments Single-Board RIO 9606 (B.3)
    (c) Analog RMC (B.3.2)
    (d) Analog Devices ADXL-335 (B.1)
    (e) Asus WL-330ge (optional)

3.1.3 Prelab Exercises

1. Appendix B: Lab Hardware, §B.2.2 (Exercises) (all)
2. Review the template project provided for the lab exercises. The code can be downloaded from the course website, under the title `Cal Climber Lab Exercises`.
   (a) The template code implements a state machine; sketch the state machine.
   (b) Does the template code use polling or interrupts to read the iRobot sensors?
   (c) A software driver for a hardware peripheral uses the `xqueue` data type. What is this data type, and for which peripheral is it used?
   Hint: review the file xqueue.h .
   (d) The `xqueue` data type encapsulates a buffer whose size must be a power of two. What is the benefit of such a constraint?
   Hint: review the file xqueue.h .
   (e) When a function returns a value of type `status_t`, what function returns `true` if this value represents an error?

3.1.4 Lab Setup

The following exercises reference files that may be downloaded from the Jensen, Lee, and Seshia website,

`http://LeeSeshia.org/lab/releases/0.05`

Be sure to completely extract all archives before opening any files, or you may receive dependency errors. Once extracted, the exercises may be found in the folder

`C/calClimber`

1. *Verify the Hardware Setup:* Your iRobot Create is interfaced with an sbRIO, an Analog RMC, and a wireless bridge – the combination is referred to as the Cal Climber. Verify the components are correctly wired, as shown in Fig. C.10 (you do not need an accelerometer for this lab).

2. *Extract the Project Files:* The project files may be downloaded from the course website,

`http://LeeSeshia.org/lab/releases/0.05/calClimberC.zip`

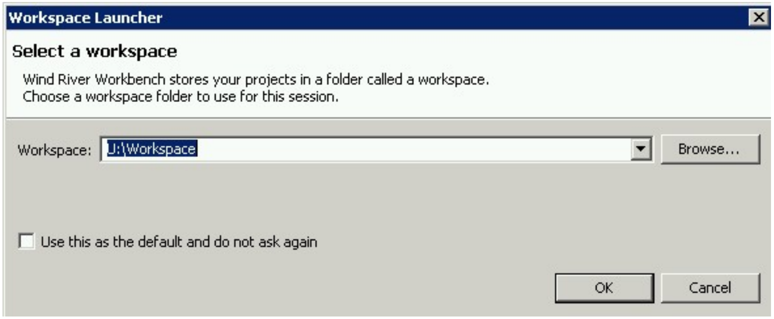
This archive includes a binary copy of the VxWorks real-time OS kernel to allow your computer to debug code running on sbRIO, and a project archive that will be imported into your Workbench workspace (no need to extract).

3. *Configure a Workspace:* From your Start menu, open "Wind River, Workbench 2.5". You will be prompted to select a workspace (Fig. 3.1. Choose a writable location *with no spaces in the path*. Workbench will create a workspace in this location if none exists already.

Once the workspace has been created, from the top menu bar select "File, Import..." to open the Import Wizard. You will import an existing project into you workspace, and browse for the file `calClimberCProject.zip` (Fig. 3.2 – 3.3).

Once imported, your workspace should show the calClimber project in the Project Navigator window.

4. *Build the Project:* Open the file calclimber.c and select from the top menu bar, "Project, Build All". This will build the template code.



Wind River Workbench stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: J:\Workspace

☐ Use this as the default and do not ask again

Figure 3.1: Selecting a workspace. The location *must not have spaces in the path*, a limitation of Workbench.

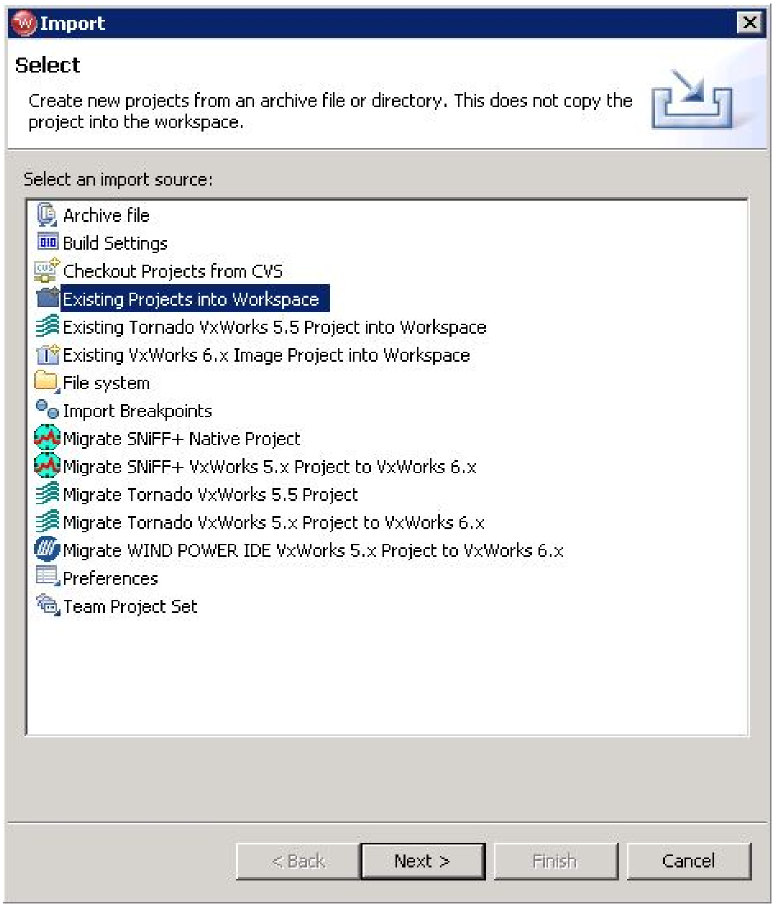
5. *Connect to your Target:* From the top menu bar, open "Target, New Connection..." to launch the Target Connection Wizard. You will connect to a Wind River VxWorks 6.x Target Server, a service running on sbRIO. In the Target Server Options dialog, enter the IP address of your target. Then browse for the VxWorks kernel image, which should extract to `C/calClimber/vxWorks_9606`. Your target options dialog should look similar to Fig. 3.4. Click next until the "Connection Summary" dialog, and change the connection name to "sbRIO". Click finish to attempt to connect to your target, and wait until the connection is 100% complete before continuing to the next step.

If you lose connection to your target, you can attempt to reconnect by right-clicking on the sbRIO target in the Target Manager pane and selecting "Connect" (Fig. 3.6).

6. *Open the Target Console:* VxWorks includes a console that displays `printf()` statements in software. To open the Target Console, first confirm you are connected to the target, then from the Target Manager pane right-click on the target and select "Target Tools, Target Console" (Fig. 3.8).

7. *Create a Debug/Run Configuration:*

   (a) From the top menu bar, select "Run, Debug...". Right-click on Kernel Task and select "New" to create a new kernel task. This will create the default action for running and debugging your task. First, select the function to call, in this case `calClimber()`, by pressing the "Browse" button and navigating to "Downloads, calClimber.out, calClimber" (Fig. **??**). If there are no .out files in the Downloads folder, your project has not yet been built. Revisit the build step above and try again.

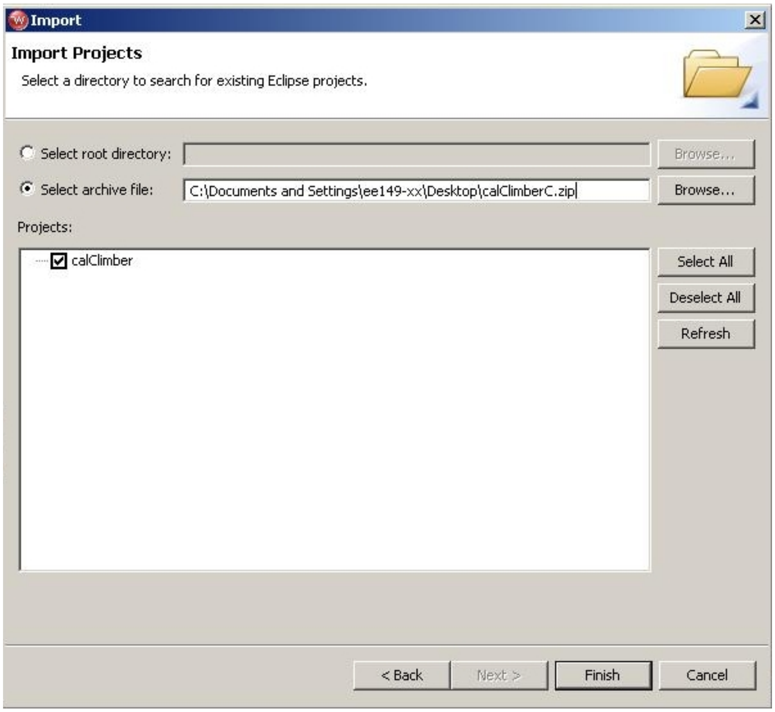


Create new projects from an archive file or directory. This does not copy the project into the workspace.

Select an import source:

- Archive file
- Build Settings
- Checkout Projects from CVS
- Existing Projects into Workspace
- Existing Tornado VxWorks 5.5 Project into Workspace
- Existing VxWorks 6.x Image Project into Workspace
- File system
- Import Breakpoints
- Migrate SNiFF+ Native Project
- Migrate SNiFF+ VxWorks 5.x Project to VxWorks 6.x
- Migrate Tornado VxWorks 5.5 Project
- Migrate Tornado VxWorks 5.x Project to VxWorks 6.x
- Migrate WIND POWER IDE VxWorks 5.x Project to VxWorks 6.x
- Preferences
- Team Project Set

Figure 3.2: Import an existing project into your workspace.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

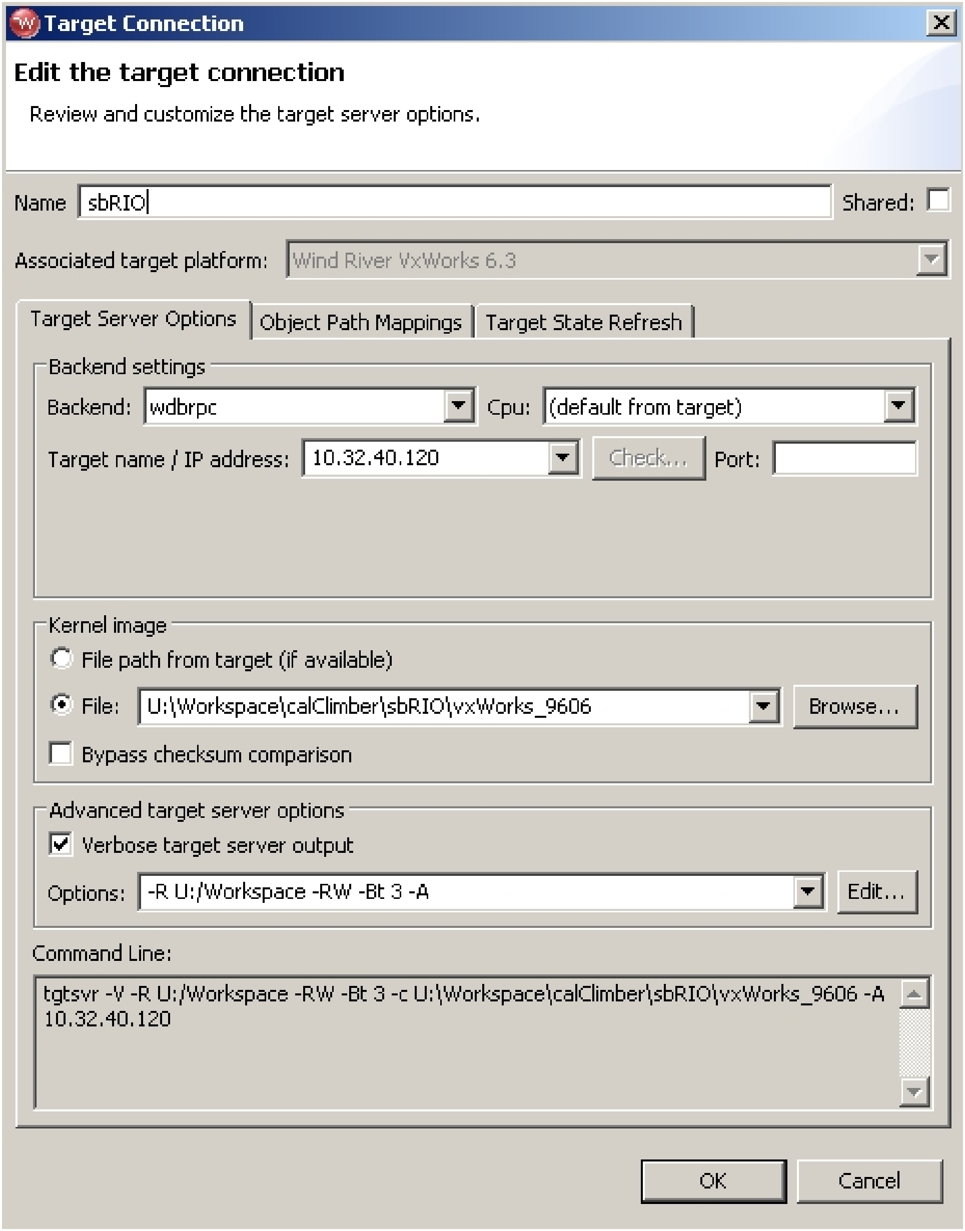
Select root directory:

Select archive file: C:\Documents and Settings\ee149-xx\Desktop\calClimberC.zip

Projects:

☑ calClimber

Figure 3.3: Browse for the project archive `calClimberCProject.zip`.



Edit the target connection

Review and customize the target server options.

Name: sbRIO   Shared: ☐

Associated target platform: Wind River VxWorks 6.3

Backend settings

Backend: wdbrpc   Cpu: (default from target)

Target name / IP address: 10.32.40.120   Port:

Kernel image

○ File path from target (if available)

● File: U:\Workspace\calClimber\sbRIO\vxWorks_9606

☐ Bypass checksum comparison

Advanced target server options

☑ Verbose target server output

Options: -R U:/Workspace -RW -Bt 3 -A

Command Line:

```
tgtsvr -V -R U:/Workspace -RW -Bt 3 -c U:\Workspace\calClimber\sbRIO\vxWorks_9606 -A
10.32.40.120
```

Figure 3.4: Target Connection Settings, reflecting the IP address of the target and the path to the VxWorks kernel image. The “name” field may not be present.

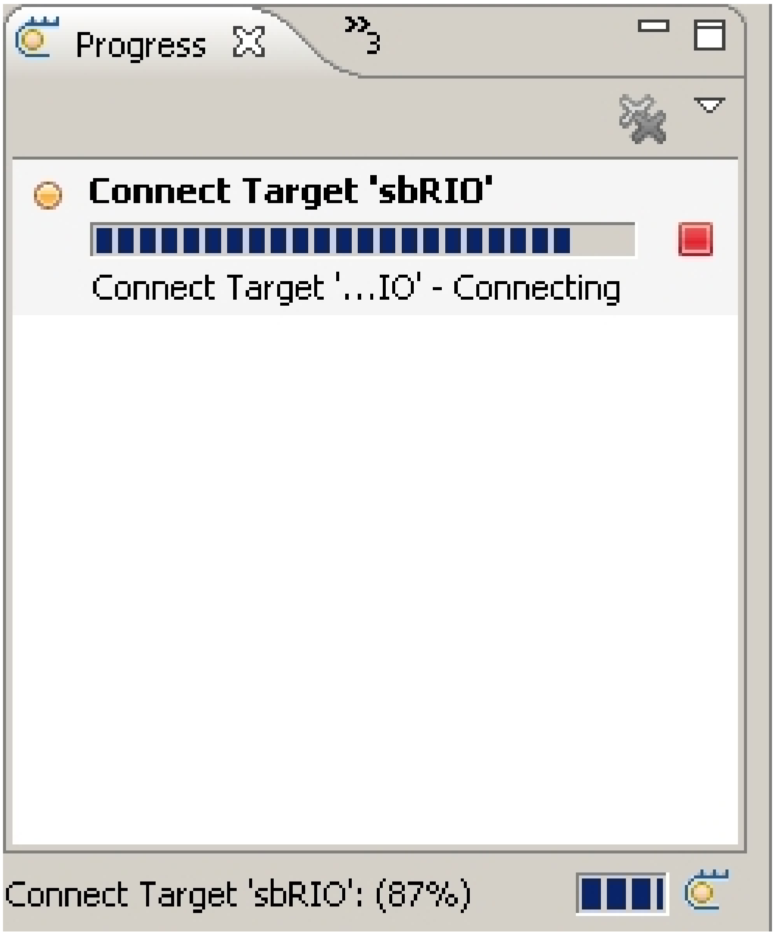

Progress
Connect Target 'sbRIO'
Connect Target '...IO' - Connecting
Connect Target 'sbRIO': (87%)

Figure 3.5: Connecting to the target.

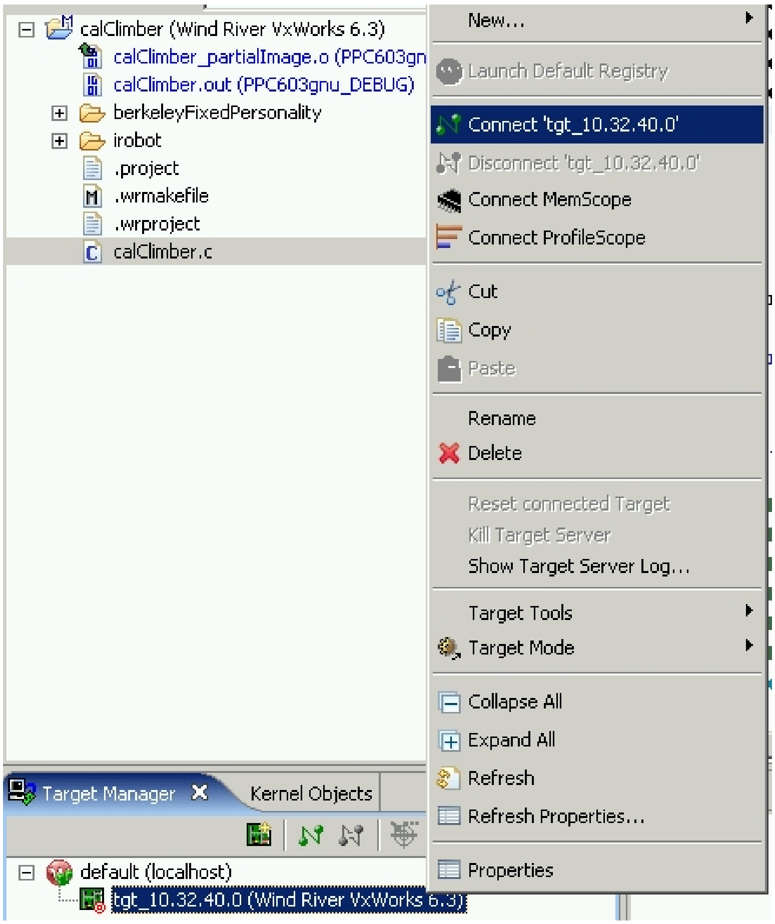


Figure 3.6: Reconnecting to the target.

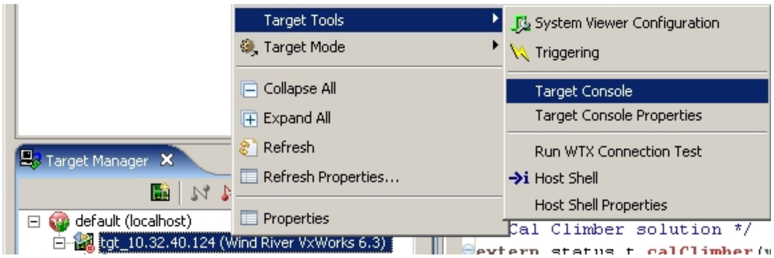
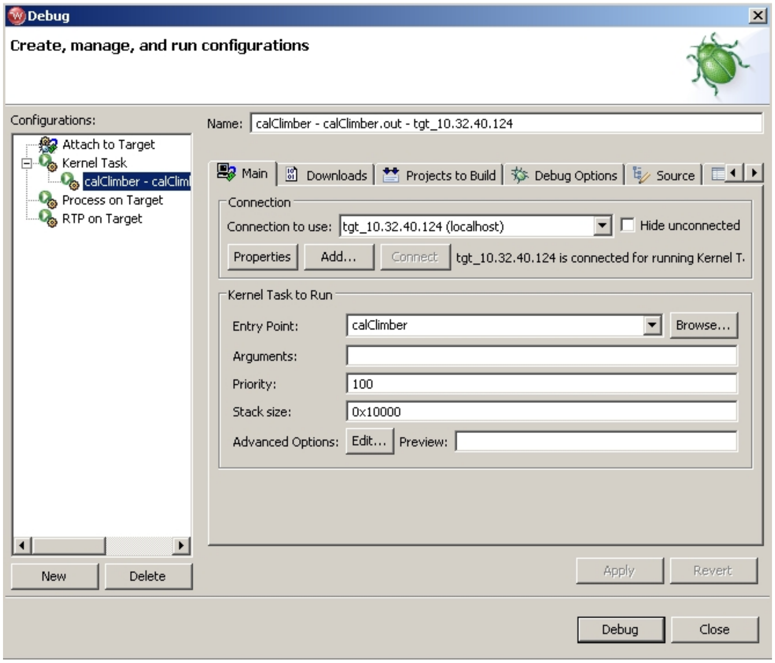


Figure 3.7: Opening the target console.



Create, manage, and run configurations

Configurations:
- Attach to Target
- Kernel Task
  - calClimber - calClim
- Process on Target
- RTP on Target

Name: calClimber - calClimber.out - tgt_10.32.40.124

Connection

Connection to use: tgt_10.32.40.124 (localhost)   ☐ Hide unconnected

tgt_10.32.40.124 is connected for running Kernel T.

Kernel Task to Run

Entry Point: calClimber

Arguments:

Priority: 100

Stack size: 0x10000

Advanced Options: Preview:

Figure 3.8: Build configuration.

(b) In the “Downloads” tab, select the build output file and click the “Edit...” button. Check both “Load Symbols to Debug Server” and “Download even when file not modified”, then click “Advanced Options...”. In the advanced options dialog, uncheck “do not unload the exiting module”. These options make it easier to start and stop debugging sessions.

(c) In the “Debug Options” tab, uncheck the box “Break on Entry” so that your code immediately begins executing when the debug button is pressed.

3.1.5 Lab Exercises

Workbench spawns your application as a task on the embedded controller, and connects this task with the debugger. Workbench is a sophisticated tool which can attach to multiple tasks simultaneously. If your application is running and you press the debug button to start a new process, you will receive the error ”Object module already loaded into memory”, indicating your code has already been downloaded to the target and there already an active or suspended debugging session. The best way to avoid this error is for your task to terminate normally, meaning the `calClimber()` function completed and returned a status. Terminating the task from the debugger will stop the execution of the code but will leave a debugging task remaining, which may prevent you from connecting to the debugger again. If this happens, simply right-click on the target in the Target Manager and select “Reset connected Target”, or alternatively, press the reset button the sbRIO itself.

The easiest way to revise code and restart debugging is for the original task to complete. The template code in your project is designed to finish execution when the ‘Advance’ button is pressed on the iRobot. When you finish with a debugging session, press the ‘Advance’ button on the robot to terminate the task and end the debug session. If you are unable to reconnect the debugger, restart the device.

1. *Navigate the iRobot:* Program the robot to drive straight when the ‘Play’ button is pressed on the top of the robot. When an obstacle is encountered, such as a cliff or an object, the robot should navigate around the object and continue in its original orientation. Keep in mind that an obstacle or cliff may be detected *during* your obstacle avoidance algorithm.

   (a) Describe (and sketch) your obstacle avoidance algorithm.

   (b) Did you follow a state machine architecture? If so, which states did you add to the default project?

2. *Share your Feedback:* what did you like about this lab, and what would you change?

3.1.6 Troubleshooting

1. **I receive the error "Object module already loaded into memory" when attempting to run on target:** A debugging session is already active on the target. Right click on the target in the Target Manager and select "Reset connected Target" to reset the controller.

To avoid this error, ensure your task exits normally before attempting a new debug session. The template code will exit when the "Advance" button is pressed on the top of the robot.

2. **I receive the error, "Unable to connect to remote target" when attempting to connect to the target.** Ensure the target is powered on and has had time to boot and connect to the wireless network. Check connectivity by using the system `ping` tool. If you are still unable to connect to the target, press the reset button on the side of the sbRIO. Be patient when connecting - it takes time to connect over wireless.

3.2 iRobot Hill Climb in C

This lab builds on the concept of programming cyber-physical systems introduced in §?? (??), and focuses on feedback-control. You will program an external embedded controller to interface with the iRobot Create robotics platform. Your goal in this lab is to implement a state machine that instructs the robot to navigate to the top of an incline while avoiding cliffs and obstacles.

3.2.1 Suggested Reading

All suggested reading from §3.1 (iRobot Navigation in C).

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

1. Introduction to Embedded Systems
   (a) Chapter 2: Continuous Dynamics, §2.4 (Feedback Control)
   (b) Draft Chapter on Sensors and Actuators.
      i. §7.1 (Models of Sensors and Actuators)
      ii. §7.2.2 (Measuring Tilt and Acceleration)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. We refer you to key topics, but you may need to reference other sections to complete these exercises.

1. Appendix B: Lab Hardware, §B.1 (Analog Devices *i*MEMS ADXL-335 Accelerometer)
2. ADXL335 Datasheet (Analog Devices, 2010)
3. ADXL335 Breakout Schematic (SparkFun, 2009)
4. Appendix B: Lab Hardware, §B.2 (iRobot Create)
5. iRobot Create Owner's Guide (iRobot, 2006b)
6. iRobot Create Open Interface (OI) Specification (iRobot, 2006a)
7. Appendix B: Lab Hardware, §B.3 (National Instruments Single-Board RIO 9606)

These documents go beyond the scope of the core concepts of these exercises, and are provided as additional resources.

1. A very straightforward treatment of differential drive robots is given in Dudek and Jenkin (2000)

3.2.2 Equipment and Software Required

1. PC computer running Microsoft Windows XP
2. Wind River Workbench 2.5 for VxWorks (Wind River Systems, 2012)
3. Cal Climber (C.3.4)
   (a) iRobot Create (B.2)
   (b) National Instruments Single-Board RIO 9606 (B.3)
   (c) Analog RMC (B.3.2)
   (d) Analog Devices ADXL-335 (B.1)
   (e) Asus WL-330ge (optional)

3.2.3 Prelab Exercises

1. Appendix B: Lab Hardware, §B.1.1 (Exercises) (all)
2. *Model physical processes:* Consider an accelerometer mounted according to the coordinate system in Fig. C.4. Let $\vec{a}_g = (a_{g_x}, a_{g_y}, a_{g_z})$ be the specific force of gravity as measured by an accelerometer. You may assume the accelerometer has been calibrated and is in units of g.
   (a) Find $\vec{a}_g$ when the robot is on level ground.
   (b) The robot is placed on a hill with inclination $\theta_I \in \left[0, \frac{\pi}{2}\right]$ in radians. Find $\vec{a}_g$ when the robot is oriented directly uphill.
   (c) The robot is placed on a hill with inclination $\theta_I \in \left[0, \frac{\pi}{2}\right]$ in radians. Find $\vec{a}_g$ when the robot is rotated θ radians counterclockwise from uphill orientation. Verify $\|\vec{a}_g\| = 1g$.
3. *Explore Joint Dynamics:* This exercise highlights the joint dynamics between physical processes and embedded control. For this exercise, consider a robot with an accelerometer mounted as shown in Fig. C.4.
   (a) An object on the surface of the Earth experiences a roughly constant gravitational force. If the robot is placed at rest in two distinct positions and orientations in space, will the accelerometer measurements in each position necessarily be equal?
   (b) In what two ways does movement of the robot influence the accelerometer measurement?
   Hint: What does an accelerometer actually measure?
   (c) One of the two components found in part 3b is undesirable when measuring tilt. What is the undesired component, and how might it be reduced or eliminated?
4. *Derive a Control Algorithm:* The iRobot Create uses a two-wheel differential drive to move. Each wheel may be actuated independently at a rate of ±500mm/s. Design a control algorithm that will navigate your robot to the top of an incline by relating wheel speed $\vec{v} = (v_L, v_R)$ to the measurement of gravity $\vec{a}_g = (a_{g_x}, a_{g_y}, a_{g_z})$.

3.2.4 Lab Setup

The following exercises reference files that may be downloaded from the Jensen, Lee, and Seshia website,

`http://LeeSeshia.org/lab/releases/0.05`

Be sure to completely extract all archives before opening any files, or you may receive dependency errors. Once extracted, the exercises may be found in the folder

`C/calClimber`

The lab setup is the same as in §3.1 (iRobot Navigation in C).

3.2.5 Lab Exercises

For the exercises below, begin with the solution you developed for §3.1 (iRobot Navigation in C).

1. *Hill Climb:* Using feedback from the accelerometer, your robot should differentiate between an incline and level ground.

On level ground, the robot should drive straight. When an obstacle is encountered, such as a cliff or an object, the robot should navigate around the object and continue in its original orientation. Keep in mind that an obstacle or cliff may be detected *during* your obstacle avoidance algorithm.

On an inline, the robot should navigate uphill, while still avoiding obstacles. You may not use cliff sensors as "bumpers" to hug the side of the incline; instead, use the accelerometer to determine uphill and employ a control algorithm to stay on heading.

   (a) Describe (and sketch) your climbing algorithm.
   (b) Did you follow a state machine architecture? If so, which states did you add?
   (c) How did you account for movement of the robot when reading from the accelerometer? Describe any algorithms or filters used.

2. *Share your Feedback:* what did you like about this lab, and what would you change?

3.2.6 Troubleshooting

Troubleshooting tips include those in §3.1 (iRobot Navigation in C).

3.3 iRobot Navigation in Statecharts

Model-based design emphasizes mathematical modeling to design, analyze, verify, and validate dynamic systems. Mathematical models are used to design, simulate, synthesize, and test cyber-physical systems, and are based on system specifications and analysis of the physical context in which the system resides. A complete model of a cyber-physical system represents the coupling of its physical processes and embedded computations. Design of these systems requires understanding of these joint dynamics, which is the focus of this lab. You will design a control algorithm using the Statecharts model of computation, verify in simulation, and deploy the solution to the iRobot Create. The goal of this lab is to implement a Statechart that instructs the robot to maintain a basic sense of orientation while avoiding obstacles.

3.3.1 Suggested Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

1. Introduction to Embedded Systems (Lee and Seshia, 2011)
   (a) Chapter 5: Composition of State Machines
   (b) Chapter 7: Embedded Processors, §7.1.1 (Microcontrollers)
   (c) Chapter 9: Input and Output, §9.1.3 (Serial Interfaces)
2. Graphical System Design with LabVIEW Statecharts (video) (Washington, 2009)
3. Developing Applications with the NI Statecharts Module (National Instruments, 2010b)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. We refer you to key topics, but you may need to reference other sections to complete these exercises.

1. Appendix B: Lab Hardware, §B.2 (iRobot Create)
2. iRobot Create Owner's Guide (iRobot, 2006b)
3. iRobot Create Open Interface (OI) Specification (iRobot, 2006a)
4. Appendix B: Lab Hardware, §B.3 (National Instruments Single-Board RIO 9606)
5. LabVIEW Statecharts Module Help (National Instruments, 2011c)

3.3.2 Equipment and Software Required

1. PC computer running Microsoft Windows XP
2. National Instruments LabVIEW 2011 (National Instruments, 2012a)
    (a) LabVIEW 2011
    (b) LabVIEW Robotics Module 2011
    (c) LabVIEW Real-Time 2011
3. Cal Climber (C.3.4)
    (a) iRobot Create (B.2)
    (b) National Instruments Single-Board RIO 9606 (B.3)
    (c) Analog RMC (B.3.2)
    (d) Analog Devices ADXL-335 (B.1)
    (e) Asus WL-330ge (optional)

3.3.3 Prelab Exercises

Complete these exercises only if you have not already done so in a previous lab.

1. Appendix B: Lab Hardware, §B.2.2 (Exercises) (all)

2. How does the Statecharts model of computation differ from finite state machines?

3. In a Statechart, what is the difference between a state and a region?

4. In LabVIEW Statecharts, what is represented by an arrow containing three open or closed cells? What does each cell represent? What does this arrow connect?

3.3.4 Lab Setup

The following exercises reference files that may be downloaded from the Jensen, Lee, and Seshia website,

`http://LeeSeshia.org/lab/releases/0.05`

Be sure to completely extract all archives before opening any files, or you may receive dependency errors. Once extracted, the exercises may be found in the folder

`LabVIEW/calClimber`

1. *Verify the Hardware Setup:* Your iRobot Create is interfaced with an sbRIO, an Analog RMC, and a wireless bridge – the combination is referred to as the Cal Climber. Verify the components are correctly wired, as shown in Fig. C.10 (you do not need an accelerometer for this lab).

2. *Extract the Project Files:* The project files may be downloaded from the course website,

   `http://LeeSeshia.org/lab/releases/0.05/calClimberLabVIEW.zip`

3. *Open the Project:* From the extracted project files, open the file `Cal Climber.lvproj` (LabVIEW Project File). It may take a few minutes to load the project. You may receive a warning that dependencies were loaded from a new path; this is normal and may be disregarded.

4. *Connect to your Target:* In the Project Explorer you will see an sbRIO target. Right click on the target, select ‘Properties’, and update the IP address with that of your robot.

3.3.5 Lab Exercises

Discrete simulations of continuous systems are approximations, and are subject to numerical error. These simulations are based on ordinary differential equations (or differential algebraic equations), that are computationally expensive to solve, and numerical accuracy is balanced against real-time performance. The scope of any simulator is limited: one simulation tool may focus on kinematics, another on electricity and magnetism, another on fluid mechanics, another on energy, another on quantum and nuclear dynamics, and so on. Some tools are dedicated to modeling the behavior of computers themselves!

A physical model of the Cal Climber is included in the LabVIEW Robotics Simulator, a tool that simulates robots, their environment, and obstacles. Simulations are rendered in 3D and in real-time. iRobot sensor packets are constructed from the state of the simulation and passed into a controller that outputs desired wheel speeds.

No simulation will be 100% accurate, and you may encounter differences between the simulated and real worlds. Specifically:

- The simulator calculates distance traveled and rotation turned by the iRobot by measuring its change in position. The real robot reports distance and angle by sensing the movement of its wheels. This yields the following differences between simulation and real performance:
  - In the simulator, distance is always unsigned. On the real robot, distance is signed. A simple way to ensure the same behavior in simulation and on the physical device is to use absolute values when calculating distance.
  - If the wheels on the real robot are slipping, the robot still reports this as distance traveled and angle turned, since the wheels are still moving. In the simulator, only actual movement of the robot is measured.
- The infrared range finders on the real robot indirectly measure distance to the ground; actually, they measure light intensity. Light intensity varies significantly between different types of materials and lighting conditions. The simulator does not model light intensity, and instead the cliff sensor signal values are directly proportional to distance. On the real robot, these values may differ based on material and lighting conditions. You may need to adjust thresholds between the simulator and the real robot.
- The simulator wraps all angles to $[-\pi, \pi]$; the real robot does not.

- Not all iRobot sensors are updated by the simulator, including charging sources available, battery life, battery temperature, song playing, etc. The following sensors are modeled by the simulator:
  - Wall signal (boolean and raw)
  - Cliff signals (boolean and raw)
  - Bumps and wheel drops (boolean)
  - Distance traveled since last sensor packet (mm) – always unsigned
  - Angle traveled since last sensor packet (deg) – always signed and wrapped to $[-\pi, \pi]$.
  - Buttons, as pressed by users on the front panel.
- The simulated accelerometer measures only gravity, and not coordinate acceleration. On the real robot, the accelerometer measures proper acceleration.

Your Statechart does not know if it is executing within a simulation or if it is executing on the physical device, so keep these differences in mind during design.

To navigate within the simulated world:

- **Click + Drag** to rotate view about the clicked point.
- **Ctrl + Click + Drag** to translate view in the plane of the screen.
- **Shift + Click + Drag** to zoom in/out.

1. *Navigate the Simulator:* In your Project Explorer, navigate to “My Computer, Simulated Cal Climber” and open Cal Climber Simulator.vi. This is the main VI for the simulator and is responsible for spawning the physics engine, calculating sensor values, and executing your Statechart. Your code should reside entirely in the Statechart – you do not need to modify any dataflow VIs.

First run Cal Climber Simulator.vi to see the default behavior of the robot. Then in your Project Explorer, navigate to “My Computer, Cal Climber State Machine, Diagram.vi”. This is where you will develop the solution to this exercise.

Program the simulated robot to drive straight when the ‘Play’ button is pressed on the front panel simulation of the iRobot. When an obstacle is encountered, such as a cliff or an object, the robot should navigate around the object and continue in its original orientation. Keep in mind that an obstacle or cliff may be detected *during* your obstacle avoidance algorithm.

(a) Provide a description and screenshot of your obstacle avoidance algorithm.

2. *Navigate the iRobot:* The Statechart you designed in the simulator is configured to deploy on your embedded target. In the Project Explorer, navigate to "sbRIO" and open Cal Climber Target.vi. This is the main VI for the sbRIO and is responsible for sampling the accelerometer, reading the iRobot sensors, and transmitting drive commands. You code should reside entirely in the Statechart – you do not need to modify any dataflow VIs, though you may wish to add a MathScript node to filter the accelerometer signal.

Run Cal Climber Target.vi to deploy your Statechart to your target.

Your robot should navigate through an obstacle course, avoiding cliffs and objects along the way.

(a) Did your Statechart require modification to work on the physical target? If so, what modifications did you make?

(b) How would you compare programming in Statecharts to programming in a traditional programming language such as C?

3. *Share your Feedback:* what did you like about this lab, and what would you change?

3.3.6 Troubleshooting

1. **I receive the error “Cal Climber.vi loaded with errors and was closed” when attempting to run on target:** We are still investigating this error and are not sure of its cause. If this happens, you may need to revert to a previously saved version of your statechart, or even the template Statechart and recreate your diagram. The Statechart is loaded from an `.lvsc` file in the project folder.

3.4 iRobot Hill Climb in Statecharts

This lab builds on the concept of model-based design for cyber-physical systems introduced in §3.3 (iRobot Navigation in Statecharts) and focuses on feedback-control. Your goal in this lab is to implement a state machine that instructs the iRobot Create to navigate to the top of an incline while avoiding cliffs and obstacles.

3.4.1 Suggested Reading

These exercises assume familiarity with suggested reading in §3.4 (iRobot Hill Climb in Statecharts).

3.4.2 Equipment and Software Required

1. PC computer running Microsoft Windows XP
2. National Instruments LabVIEW 2011 (National Instruments, 2012a)
   (a) LabVIEW 2011
   (b) LabVIEW Robotics Module 2011
   (c) LabVIEW Real-Time 2011
3. Cal Climber (C.3.4)
   (a) iRobot Create (B.2)
   (b) National Instruments Single-Board RIO 9606 (B.3)
   (c) Analog RMC (B.3.2)
   (d) Analog Devices ADXL-335 (B.1)
   (e) Asus WL-330ge (optional)

3.4.3 Prelab Exercises

There are no prelab exercises for this lab.

3.4.4 Lab Setup

The following exercises reference files that may be downloaded from the Jensen, Lee, and Seshia website,

`http://LeeSeshia.org/lab/releases/0.05`

Be sure to completely extract all archives before opening any files, or you may receive dependency errors. Once extracted, the exercises may be found in the folder

`LabVIEW/calClimber`

The lab setup is the same as in §3.3 (iRobot Navigation in Statecharts).

3.4.5 Lab Exercises

For the exercises below, begin with the solution you developed for §3.3 (iRobot Navigation in Statecharts); be sure to review the prefix to the exercises section, as it underscores important differences between the simulator and the real robot.

1. *Hill Climb in the Simulator:* Using feedback from the accelerometer, modify your Statechart to differentiate between an incline and level ground.

On level ground, the simulated robot should drive straight. When an obstacle is encountered, such as a cliff or an object, the robot should navigate around the object and continue in its original orientation. Keep in mind that an obstacle or cliff may be detected *during* your obstacle avoidance algorithm.

On an inline, the simulated robot should navigate uphill, while still avoiding obstacles. You may not use cliff sensors as "bumpers" to hug the side of the incline; instead, use the accelerometer to determine uphill and employ a control algorithm to stay on heading.

(a) Provide a description and screenshot of your climbing algorithm.

2. *Hill Climb on the iRobot:* Deploy your Statechart to the real target. Your robot should navigate through an obstacle course and navigate to the top of an incline, avoiding cliffs and objects along the way.

(a) Did your Statechart require modification to work on the physical target? If so, what modifications did you make?

3. *Share your Feedback:* what did you like about this lab, and what would you change?

3.4.6 Troubleshooting

Troubleshooting tips include those in §3.3 (iRobot Navigation in Statecharts).

4 Projects

Contents

The exercises in this chapter guide you through basic skills in project management, a critical component of any successful capstone design project. Additionally, a series of open-ended projects are proposed as starting points for a significant design project.

Contributor: Christopher X. Brooks, University of California, Berkeley.

4.1 Project Management

A capstone design project is a significant team undertaking, and can benefit from some simple project management. Well-managed projects are less stressful and ultimately more successful. Some useful project management tasks include:

1. Specify the project by submitting a one-page charter.
2. Once a project charter has been reviewed and approved, submit a project plan of action, a timeline for milestones, and a division of responsibilities for team members.
3. Indicate core concepts addressed by the project, such as concurrency, modeling of physical dynamics, reliable real-time behavior, modal behavior governed by finite state machines coupled with formal analysis, real-time networks, simulation strategies, and design methodologies for embedded systems design.
4. Pair with a mentor with relevant experience who is a professor, graduate student, researcher, or industry professional.
5. Perform weekly progress checkpoints, which may alternate between in-class presentations and one-page milestone reports comparing progress to the original project charter.
6. At the project halfway point, host live demonstrations during a department open-house, which facilitates functional milestones.
7. Periodically submit peer evaluation forms to identify any issues with a particular team member.

4.1.1 Suggested Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

1. Project Charter Instructions (Brooks, 2008)

2. Tortoise SVN (Kung et al., 2011)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. We refer you to key topics, but you may need to reference other sections to complete these exercises.

1. Wikipedia
   (a) Project Charter (Wikipedia, 2012b)
   (b) Milestone (Project Management) (Wikipedia, 2012a)
   (c) Work Breakdown Structure (Wikipedia, 2012d)
2. EECS Instructional Public Documentation - SVN Help (Brooks, 2012)

4.1.2 Equipment and Software Required

1. Desktop computer
2. Word processor
3. Tortoise SVN (Tortoise SVN Team, 2012)

4.1.3 Prelab Exercises

1. What are two advantages of using a version control system such as SVN?
2. Alice and Bob both update their working copies at the beginning of the day. They start editing the same plain-text file. Alice commits her changes in the afternoon. What happens when Bob tries to commit his changes at the end of the day? What should Bob do?
3. What is the SVN checkout URL for your group (your group number is on the Google spreadsheet)?
4. How long should a project charter be?
5. Why is it important to have a *brief* charter?
6. Suppose you are writing the work breakdown to design an automobile, which has subsystems such as steering, horn, drivetrain, transmission, motor, etc. Give an example of a breakdown which would
   (a) Violate the 100% rule.
   (b) Violate the Mutually Exclusive rule
7. Make a short list of skills / experience / interests you think would be most relevant for your project. (i.e. Microcontroller coding, Circuit design, Project Management, etc.) Next to each, give an example project or class you took which demonstrates that skill.
8. If you know you will be using a mentored project from the website, communicate with them *before this lab*, asking for whatever details you need to know to complete the charter & schedule. i.e. you will need to know what components need to be designed, whether there is hardware tasks involved, if there are deadlines outside of this class, and other considerations. You should also schedule regular meetings with them for them to track your progress. List all such components, tasks, and deadlines.

4.1.4 Lab Setup

TortoiseSVN is a *frontend* for the program SVN, a version-control system. TortoiseSVN provides handy right-click menus for updating, committing and reviewing your files.

1. *Check out a Repository:* To checkout your repository, create a new directory where you want the files (preferably one on a network drive). Right-click on the new directory and select “SVN Checkout...”. In “URL Repository” enter the URL of your repository. When a module is checked out, notice a check mark appears next to the folder indicating your local copy is up-to-date.

2. *Add Files to your Repository:* Copy or move the files from your project charter and schedule into this newly created folder. Select both and right-click on them. Select “Tortoise SVN → Add...”. Notice there are now plus-signs next to the files. This means that this file is to be added to the repository on the next commit.

Go up a directory. The folder containing your local copy should now have a red exclamation point indicating your local copy is out-of-date. Right-click on the folder and select “SVN Commit...”. Enter a message such as “Added project charter & schedule”. *Although not strictly enforced, it is good practice to always provide a commit message for other users and for remembering what you did in the future* . The folder should now turn green again.

Go back into the directory. Your project charter and schedule should now have green checkmarks indicating they are up-to-date. Try changing one of them– you should get a red exclamation point indicating local changes are out-of-date. You can commit these changes in the same way as before, but suppose you made a mistake and you want to undo your changes. Right-click on the file and select “Tortoise SVN → Revert...”. If you press “OK”, the latest version in the repository will overwrite your copy. You can also revert to several versions behind if you want.

Note that because your files are probably in MS Word or some other binary format, then you cannot *diff* them easily. The *diff* feature works on plain-text files, such as C source files, and highlights the changes, line-by-line, that you changed between two revisions. This is essential for finding out which lines of code broke your program. For plain-text files, you can also handle *merges* which happen if two people tried to change the same lines– you will have to pick which one is used. For binary files, to “merge” you must simply pick one entire file or the other.

You will probably want to have every member set up their respective checkouts from your repository. This will allow you to work on the code somewhat independently, but with a central master copy. You should get in the habit of committing your changes at least once a day (like the CTRL-S of code development). Your TAs and mentors will also be using your repository to be familiar with your code.

3. *(optional) Install SVN on Another Computer:* This section is for your information in case you want to set this up on your own computer.

Windows : Installation is fairly easy– simply download and install TortoiseSVN from `http://tortoisesvn.net/downloads.html`, making sure to get 32-bit or 64-bit based on what your OS is. On Windows 7, there may be problems with the icons showing properly. Therefore, you will need to use the "check for modifications" item in TortiseSVN's right-click menu.

Linux : The "svn" command-line program should be available in most package repositories. For example, on Ubuntu systems, you can use "sudo apt-get install subversion". Refer to the man-page or online documentation for information on how to use the command-line program. There are also frontends available, such as kdesvn ("sudo apt-get install kdesvn").

Mac : Mac OS X 10.7 may already have Subversion installed. To test this, start up a Terminal window and run "svn help". If svn is not found, then install XCode from `http://developer.apple.com/xcode`. Note that XCode is roughly a 1 gigabyte download. Apple is charging $5 for XCode 4, but XCode 3 will also work.

Unfortunately for the Mac, the free GUI Subversion options are fairly limited. SCPlugin seems to be the best option out there, coming from the makers of subversion. It can be found at `http://scplugin.tigris.org/`.

4.1.5 Lab Exercises

The exercises here guide you in the creation of a project charter. Your charter serves as the lab report for these exercises. Do not enumerate the questions below in your charter, but instead make sure that each question is answered somewhere in your project charter.

1. *Team Member and Project Selection:* During this lab (your choice – before or after your charter is written) discuss with your lab TA that all skills requirements for your project are satisfied by your group members. If you have a large (4 or 5) group, be prepared to justify why the extra manpower is needed.

   (a) Who are your project members?

   (b) Give a one or two sentence description of your project.

2. *Plan your Project:* Follow Project Charter Instructions (Brooks, 2008) to create a project charter.

Ensure that you are reasonable in your objectives and deliverables, especially considering the experience of your group. i.e. Do not make a circuit board a major deliverable if no group member has any circuit design experience. On the same token, the scope of the project should match the size of the group and the goal be to produce an interesting and impressive product.

Your charter should include:

(a) Overview

(b) Approach

(c) Objectives

(d) Major Deliverables

(e) Constraints

(f) Risk and Feasibility

Your charter should be one (1) page. The point is succinctly summarize your project for a busy manager.

Each group should submit one charter.

3. *Set Milestones and a Schedule:* Working backwards from the completion due date, determine when tasks need to be done to complete the project on time. *Ensure you are allowing adequate time for debugging – integrating disjoint components from several people can take a week or more.* Be sure to consider events such as Spring Break, Finals Week, etc.. Milestones should indicate which team member is responsible and be no more than one week apart.

Be sure to include non-technical tasks such as practicing the presentation or writing the report. Also include a plan to meet with your mentor, preferably once a week (you may want to organize your milestones to land on these dates). Be sure to include vacations like Spring Break.

Do not hesitate to be painfully specific. Through the duration of this project, you should submit milestone reports to track your progress. Certainly, the milestones could change during the course of the project as you learn and debug things.

4. *Set a Work Breakdown Structure:* Generate a Work Breakdown Structure (WBS) for your project, being careful to satisfy the 100% and mutual exclusion rules. The diagram should show dependencies between milestones and which tasks can be performed in parallel. For simpler projects, this could be very linear, but try to divide the tasks so as much as possible can be done in parallel so work can be split between people. Ensure that the WBS answers when a milestone will be finished and who is responsible for it finishing.

5. *Check your Charter into SVN.*

6. *Individual Writeup:* In addition to the project charter, each team member should write an individual report which answers the following questions:

(a) What is the critical path on your WBS? How long will it take to complete?

(b) What are the amounts time you have allotted to:

   i. Design
   ii. Development (i.e. new stuff being written)
   iii. Debugging
   iv. Other tasks (such as writing presentations or reports)

(c) How have you designed your schedule adjust for unforeseen delays or early completions?

(d) When have you regularly scheduled to meet with your mentor?

A

Lab Software

Contents

This appendix describes software used in the lab exercises in this text and provides citations with links to detailed documentation. The labs include hands-on experience at three levels of abstraction. At the highest level, we use LabVIEW to introduce students to model-based design. One level down from that, we use the C programming language and a real-time operating system (**RTOS**). One level down from that, we use C on a **bare-iron** microprocessor (a microprocessor without an operating system).

A.1 LabVIEW

To gain experience with model-based design, we use **National Instruments LabVIEW** (Laboratory Virtual Instrument Engineering Workbench) (Kodosky et al., 1991). LabVIEW is a graphical design environment for scientists and engineers. LabVIEW applications can run on desktop computers or embedded controllers, and are most commonly designed to interface with sensors, actuators, instruments, data acquisition devices, and other computers and embedded devices.

A LabVIEW application is called a **virtual instrument** (**VI**). A VI consists of a **front panel** and a **block diagram**. The front panel provides an interface for setting parameters, executing a VI, and viewing results. The block diagram defines what the VI does when it executes. LabVIEW block diagrams use a particular form of a dataflow model of computation (MoC), as described in Lee and Seshia (2011), Chapter 6: Concurrent Models of Computation, §6.3 (Dataflow Models of Computation).

Useful references:

1. LabVIEW product page (National Instruments, 2012a)
2. LabVIEW 2011 Help (National Instruments, 2011a)
3. Getting Started with LabVIEW (National Instruments, 2010c)
4. LabVIEW Quick Reference Card (National Instruments, 2010d)

Related reading:

1. Introduction to Embedded Systems (Lee and Seshia, 2011)
   (a) Chapter 6: Concurrent Models of Computation, §6.3.4 (Structured Dataflow)

A.1.1 LabVIEW Control, Design, & Simulation Module

The **LabVIEW Control, Design, and Simulation Module** adds a continuous time model of computation to LabVIEW, along with tools for designing and implementing feedback control algorithms.

Useful references:

1. LabVIEW Control Design Help (National Instruments, 2011b)

Related reading:

1. Lee and Seshia (2011)
   (a) Chapter 2: Continuous Dynamics, §2.2 (Actor Models)
   (b) Chapter 6: Concurrent Models of Computation, §6.4.3 (Continuous-Time Systems)

A.1.2 LabVIEW MathScript

LabVIEW MathScript is an add-on module for the LabVIEW that adds text-based math into the graphical development environment of LabVIEW. Using this native solution for text-based math, you can combine graphical and textual programming within LabVIEW. (National Instruments, 2011g)

Useful references:

1. LabVIEW MathScript RT product page (National Instruments, 2012b)
2. What is the LabVIEW MathScript RT Module? (National Instruments, 2011g)
3. Developing Algorithms Using LabVIEW MathScript RT Module (National Instruments, 2010a)
4. MathScript RT Module Functions (National Instruments, 2009)

A.1.3 LabVIEW Real-Time

LabVIEW Real-Time Module is an add-on to LabVIEW that enables applications to be deployed to embedded targets running a real-time operating system (RTOS).

A.1.4 LabVIEW Statecharts Module

The **LabVIEW Statecharts Module** adds the Statecharts (Harel, 1987) model of computation to LabVIEW.

Useful references:

1. Graphical System Design with LabVIEW Statecharts (Washington, 2009)
2. Developing Applications with the NI Statecharts Module (National Instruments, 2010b)
3. LabVIEW Statecharts Module Help (National Instruments, 2011c)

Related reading:

1. Lee and Seshia (2011)
    (a) Chapter 3: Discrete Dynamics, §3.3 (Finite-State Machines)
    (b) Chapter 3: Discrete Dynamics, §3.4 (Extended State Machines)
    (c) Chapter 5: Composition of State Machines

B Lab Hardware

Contents

This appendix describes hardware used in the lab exercises in this text and provides citations with links to detailed documentation.

B.1 Analog Devices *i*MEMS ADXL-335 Accelerometer

Useful references:

1. ADXL-335 Datasheet (Analog Devices, 2010)
2. ADXL-335 product page (Analog Devices, 2011)

Related reading:

1. Introduction to Embedded Systems (Lee and Seshia, 2011)
   (a) Draft Chapter on Sensors and Actuators.
      i. §7.1 (Models of Sensors and Actuators)
      ii. §7.2.2 (Measuring Tilt and Acceleration)
2. Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors (Ozyagcilar, 2011)
3. Using an Accelerometer for Inclination Sensing (Fisher, 2010)

B.1.1 Exercises

1. Use the linear model for digital sensors to model the interaction between an ADXL335 analog accelerometer and an ADC:

(a) Given an input voltage $V_{ss} = 3.35\text{V}$, what is the sensitivity and bias of the accelerometer? Be sure to include units.

(b) The accelerometer is connected to a 12-bit ADC whose input range is $[0\text{V}, 5\text{V}]$. What is the sensitivity and bias of the accelerometer as seen in software that reads the ADC?

B.2 iRobot Create

The **iRobot Create** is an off-the-shelf platform capable of driving, sensing bumps and cliffs, executing simple scripts, and communicating with an external embedded controller. The iRobot Create is a complete robot development kit that can be controlled without consideration of mechanical assembly or machine code. An internal microcontroller samples its sensors, drives its actuators, and communicates with external devices over a proprietary serial interface called the iRoobt Open Interface (OI) standard (iRobot, 2006a). The iRobot Create OI defines the electronic and software interface for controlling the robot.

Useful references:

1. iRobot Create Owner's Guide (iRobot, 2006b)
2. iRobot Create Open Interface (OI) Specification (iRobot, 2006a)

Related reading:

1. Lee and Seshia (2011)
   (a) Chapter 7: Embedded Processors, §7.1.1 (Microcontrollers)
   (b) Chapter 9: Input and Output, §9.1.3 (Serial Interfaces)

B.2.1 Charging

The iRobot can be finicky about its state while charging; there are some modes in which the robot will be connected to a charging source but will not actually charge. When correctly charging, the iRobot Power LED should be red, slowly fading on and off. Solid red or flashing red indicate a charging error. Solid green is, unfortunately ambiguous: it may indicate charging is complete, or it may indicate the robot is powered on and running. You may disambiguate this by removing the charging source: if the robot was powered and running, the power LED will remain solid green. If the robot is fully charged, the power LED will turn off. If you have difficulty charging your robot, first unplug any charging sources, power off the robot, and power off any external devices such as the wireless router. Then connect the charging source. Charge often and be sure to verify your robot is in charging mode when you leave the lab.

B.2.2 Exercises

For exercises asking for a sequence of bytes, use hexidecimal (`0xXX`) notation for numbers.

1. How many sensors are on the bottom of the iRobot Create? Does the placement of the sensors result in equal sensing capability when the robot is driving forward and backward?

2. What sequence of bytes should be sent to the iRobot Create UART port to command the robot to drive straight at a velocity of 300 mm/s?

3. What sequence of bytes should be sent to the iRobot Create UART port to request a single sensor packet containing all sensors?

4. What sequence of bytes should be sent to the iRobot Create to enable a continuous stream of a single packet containing all sensors?

5. After the iRobot Create has received the sequence of bytes found in Exercise 4, what sequence of bytes will the robot transmit periodically? You may represent a sensor packet as `[Packet Name (N)]` where N is the length of the packet, and checksum as `[Checksum]`.

B.3 National Instruments Single-Board RIO 9606

The National Instruments Single-Board RIO (**sbRIO**) 9606 is an embedded microcontroller with a multiprocessor architecture. It consists of a Freescale PowerPC processor and a Xilinx field-programmable gate array (**FPGA**). The PowerPC is a single-chip processor with a fixed set of peripherals; it executes software according to a fixed instruction set arcitecture (**ISA**) developed by IBM.

Useful references:

1. NI sbRIO-9605/9606 OEM Operating Instructions and Specifications (National Instruments, 2011d)
2. NI sbRIO-9606 Product Page (National Instruments, 2011e)
3. NI Single-Board RIO Setup and Services (National Instruments, 2011f)

Related reading:

1. Lee and Seshia (2011)
   (a) Chapter 7: Embedded Processors, §7.1.1 (Microcontrollers)
   (b) Chapter 7: Embedded Processors, §7.2.4 (Multicore Architectures)
   (c) Chapter 9: Input and Output, §9.3 (The Analog/Digital Interface)

B.3.1 A Reconfigurable Architecture

The Xilinx FPGA is a reconfigurable processor comprised of logic units, memory, and other fundamental building blocks that may be reconfigured *at the hardware level*. When configured appropriately, an FPGA can implement hardware peripherals such as communication buses, PWM generators, quadrature encoder interfaces, signal processing algorithms, video rending and decoding. FPGAs can be configured to implement a microprocessor and ISA that can be easily modified with new peripherals and functionality. A microprocessor implemented in an FPGA is known as a **soft-core processor**. Soft-core processors are usually slower than a custom silicon processor with equivalent functionality, but they are nonetheless attractive because they are customizable. Often, multiple soft-core processors may be fit onto a single FPGA to create a multicore processor.

An **FPGA fixed personality** is a configuration of an FPGA that is meant to be distributed, and generally is not modified. A fixed personality is a compiled, binary file that can be distributed without source code, much like a dynamic-link library (DLL) or shared object

(SO) library. A multiplicity of fixed personalities can be used to rapidly reconfigure an FPGA, allowing a single hardware device to serve a broad range of applications.

B.3.2 Analog RMC

The sbRIO-9606 uses a high-density mezzanine interface to expose digital IO. An add-on modules with this interface is referred to as a **RIO Mezzanine Card** (RMC). We have created a custom RMC for this lab that adds analog input capabilities. We document its use here, and provide detailed schematics and layout on the course website,

`http://LeeSeshia.org/lab/releases/0.05/setup/sbrio9606AnalogModule.zip`

The analog RMC exposes 48 pins of digital I/O (DIO) and 16 pins of analog input (AI) (Fig. B.1). DIO lines connect directly to the FPGA on the sbRIO; analog lines are sampled by an analog-to-digital converter (ADC) which communicates with the FPGA on the sbRIO over a serial protocol. The ADC has 16 input channels with 12 bits of resolution per channel, and is made by Texas Instruments (Texas Instruments, 2010). The ADC can only sample one channel at a time, so multiplexing should be used to sample multiple the channels. A conversion for a single channel samples and quantizes the input voltage range of [0V, 5V] into 12 bits. Each conversion takes 16 microseconds to complete.

Interface to C and LabVIEW on the PowerPC

The Berkeley fixed personality (C.2.1) provides a hardware interface to the ADC on the Analog RMC and a software interface for both C and LabVIEW code running on the PowerPC, as well as the MicroBlaze processor core.

Interface to MicroBlaze

The MicroBlaze fixed personality (B.5) provides a hardware interface to the ADC on the Analog RMC (Fig. B.2) and a software interface for C code running on MicroBlaze. The software interface uses memory-mapped registers to communicate with the ADC, shown in Fig. B.3 – B.8.

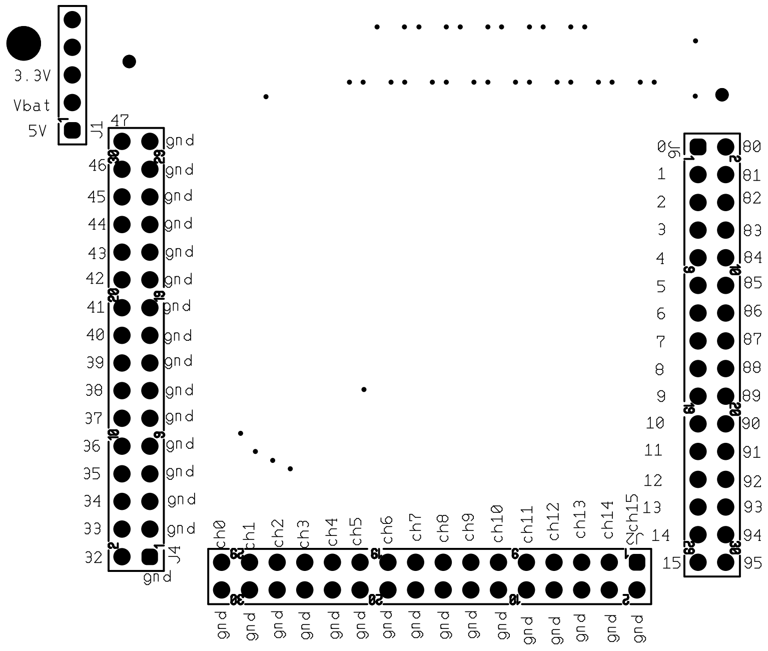


Figure B.1: Pinouts of the Analog RMC.

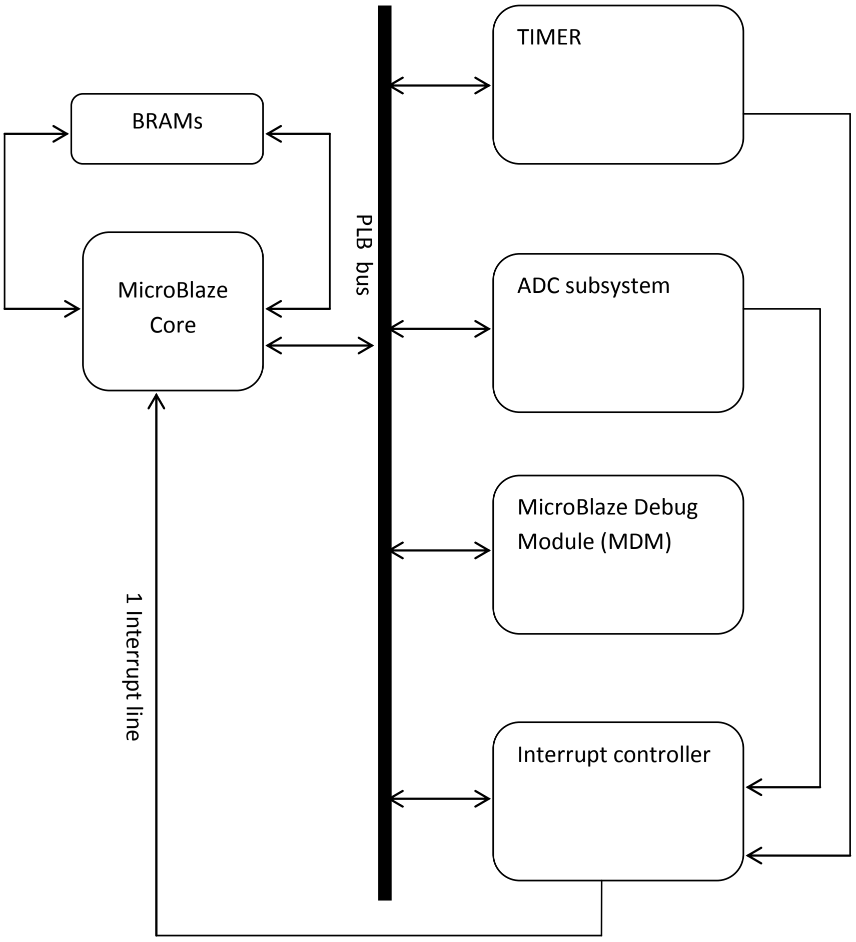


Figure B.2: Conceptual block diagram of the ADC interface to MicroBlaze.

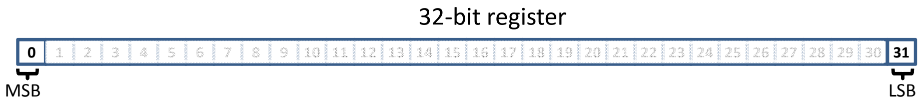
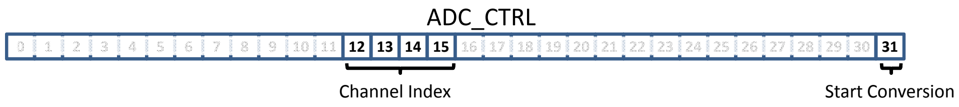
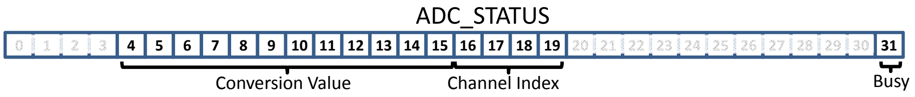


Figure B.3: Standard diagram for an ADC register on MicroBlaze. MicroBlaze is a 32-bit big-endian architecture. MicroBlaze documentation uses reverse-bit numbering (an artifact from IBM PLB technology), with the least significant bit (LSB) at bit 31 and the most significant bit (MSB) at bit 0. The LSB can be isolated with the mask 0x0000 0001, and the MSB can be isolated with the mast 0x8000 0000.



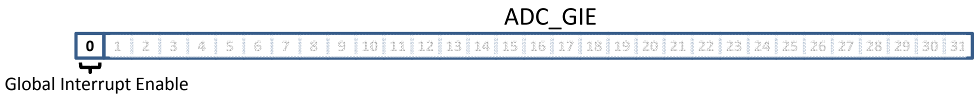
Bit	Field	Access	Values
12 – 15	Channel Index	Write	Channel 0 – 15
31	Start Conversion	Write	0: Reset 1: Start conversion

Figure B.4: ADC_CTRL register. When the Start Conversion bit changes from 0 to 1 (a rising-edge transition), the ADC begins a conversion on the channel specified in the Channel Index field. When the ADC_STATUS register is read, a 0 is written to the Start Conversion field.



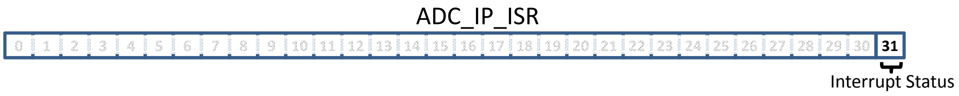
Bit	Field	Access	Values
4 – 15	Conversion Result	Read	0 – 4095
16 – 19	Channel Index	Read	0 – 15
31	ADC Busy	Read	0: Inactive 1: Conversion in progress

Figure B.5: ADC Status (ADC_STATUS) register. When an ADC conversion is in progress, the ADC Busy field is set to 1. When an ADC conversion is complete, the ADC Busy field is set to 0, Channel Index field holds the index of the ADC channel sampled, and Conversion Result field holds the result of the conversion. Reading from this register will write a value of 0 to the ADC_CTRL Start Conversion field.



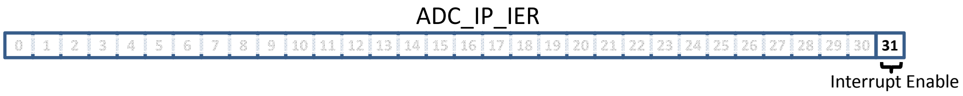
Bit	Field	Access	Values
0	Global Interrupt Enable	Write	0: Disable 1: Enable

Figure B.6: ADC Global Interrupt Enable (ADC_GIE) register. Writing a 0 to the Global Interrupt Enable field causes the master interrupt controller to ignore ADC interrupts, writing a 1 causes the master interrupt controller to listen and respond to ADC interrupts.



Bit	Field	Access	Values
31	Interrupt Status	Read	0: Inactive 1: Active
		Toggle	0: No effect 1: Toggle

Figure B.7: ADC Interrupt Status (ADC_IP_ISR) register. Reading a 0 from the Interrupt Status field indicates no interrupt is active, and reading a 1 indicates an interrupt is active. Writing a 0 to the Interrupt Status Field has no effect, and writing a 1 toggles interrupt status.



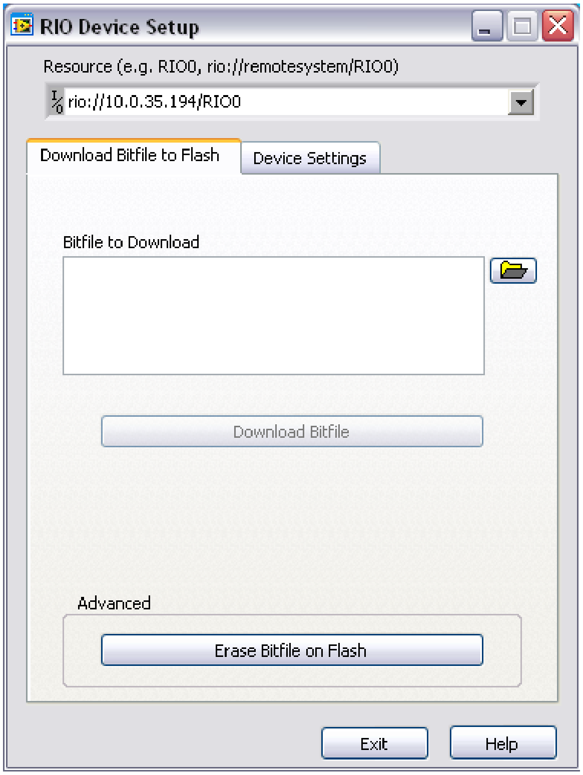
Bit	Field	Access	Values
31	Interrupt Enable	Write	0: Disable 1: Enable

Figure B.8: ADC Interrupt Enable (ADC_IP_IER) register. Writing a 1 to the Interrupt Enable field enables interrupt output, and writing a 0 disables it.

B.3.3 RIO Device Setup

The National Instruments RIO Imaging Tool erases and reprograms the FPGA of an sbRIO (or other RIO device). The tool downloads a bitfile directly to the flash memory of the FPGA, and optionally configures it to load on startup. This is the simplest way of deploying an FPGA fixed personality.

The RIO Device Setup tool is installed with the NI-RIO device drivers and may be launched from the Windows Start menu under "All Programs, National Instruments, NI-RIO, RIO Device Setup":



Resource (e.g. RIO0, rio://remotesystem/RIO0)
rio://10.0.35.194/RIO0

Figure B.9: RIO Device Setup.

Note that RIO Device Setup does not program the flash memory of the PowerPC.

B.3.4 RIO Imaging Tool

The RIO Imaging Tool erases and reprograms the filesystem on sbRIO (or other RIO device), restoring the device to a known state. This is especially helpful in a lab environment where multiple teams may use the same hardware and potentially leave it in an unknown state. The tool downloads a binary image directly to the flash memory of the controller. RIO Imaging Tool is included in the laboratory files,

```
cpsLab/tools/RIO Imaging Tool/rioimage.exe
```

. If an image deployed to the filesystem of sbRIO includes an FPGA bitfile configured to load at startup, this will have the same effect as using the RIO Device Setup to deploy an FPGA fixed personality.

B.3.5 Exercises

1. What software tool can be used to discover an sbRIO (or other National Instruments devices) on a local network?

2. What is the CPU clock frequency of the PowerPC processor on sbRIO-9606?

3. In this exercise, you will explore the electrical characteristics of the controller. These are important, since exceeding tolerances may damage or destroy your controller.

   (a) What is the input voltage range for the controller?
   (b) What is the maximum current that may be sourced from a single DIO line?
   (c) What is the total maximum current that may be sourced from all DIO lines?
   (d) When a DIO line is configured as a sourcing output, what is its ‘high’ output voltage?
   (e) When a DIO line is configured as an input, what is the maximum input voltage that should be applied?

4. Can more than one ADC channel be sampled at a time?

5. Consider the following line of code, which maps `ADC_STATUS` to a memory-mapped register:

```
#define ADC_STATUS (*((volatile unsigned long *)(XPAR_ADC_0_BASEADDR+4)))
```

(a) What is the effect of the **`volatile`** keyword?

(b) What is the effect of the typecast `(`**`volatile unsigned long`**` *)` ?

(c) What is the effect of the outer `*` operand?

6. In this exercise, you will explore the memory-mapped register interface to the ADC on the Analog RMC.

(a) Which register must be configured for the master interrupt controller on MicroBlaze to handle ADC interrupts?

(b) Which register must be configured for ADC interrupts to be generated?

(c) To which register should an ISR write to clear the interrupt status?

(d) Which register holds the result of an ADC conversion?

(e) Write a single C statement to trigger an ADC conversion on channel 5.

(f) Write a single C statement to block until an ADC conversion is complete.

(g) Write a single C statement to read the result of an ADC conversion into the variable `adcConversionResult`, and another statement to read the index of the channel associated with the conversion into `adcChannelIndex`.

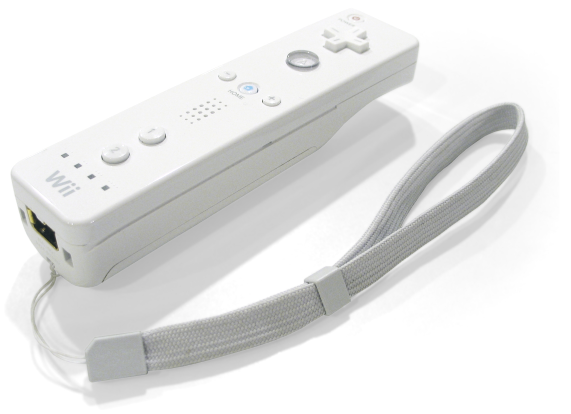


Figure B.10: Nintendo Wii Remote (Orlando, 2010).

B.4 Nintendo Wii Remote

The **Nintendo Wii Remote** (WiiMote) (Fig. B.10) is a handheld wireless game controller with motion sensing capability. It is an embedded system composed of sensors, actuators, a wireless radio, and a microcontroller. Its sensors consist of a three-axis analog accelerometer, an infrared camera, and momentary buttons; its actuators consist of a speaker, a DC motor (for "rumble" functionality), and LEDs (light-emitting diodes). The WiiMote is a simple, popular, and widely available embedded system, and its wireless interface allows it to interact with a desktop computer.

Useful references:

1. Wii Remote product page (Nintendo, 2011)
2. Wii Remote on Wikipedia (Wikipedia, 2012c)
3. WiiBrew WiiMote wiki (WiiBrew, 2011)
4. Bluetooth HID Profile (Bluetooth Special Interest Group, 2003)

Related reading:

1. Lee and Seshia (2011)
   (a) Chapter 7: Embedded Processors, §7.1.1 (Microcontrollers)
   (b) Chapter 9: Input and Output, §9.3 (The Analog/Digital Interface)

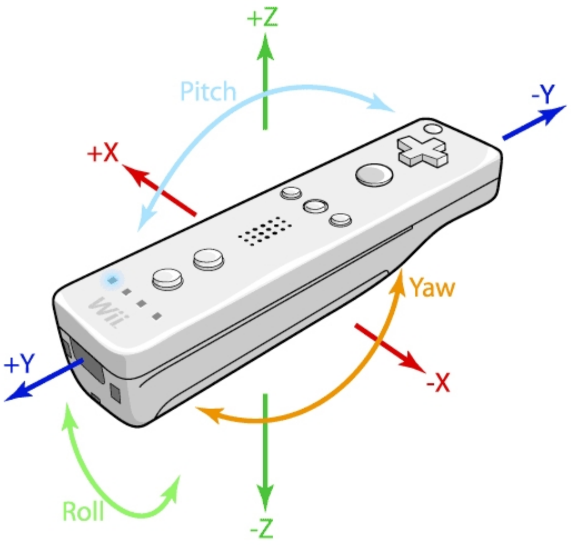


Figure B.11: Nintendo Wii Remote accelerometer axes (Stoops, 2008).

2. Using an Accelerometer for Inclination Sensing (Fisher, 2010)
3. Implementing a Tilt-Compensated eCompass (Ozyagcilar, 2011)

B.4.1 Wireless Interface

The Bluetooth **HID** (Human Interface Device) profile defines a communication protocol used by the WiiMote (Bluetooth Special Interest Group, 2003). Each command begins with a 1-byte Transaction Header, the first nibble of which corresponds to the Transaction Type, and the second to a Parameter. The header is followed by a device-specific Report ID, and sometimes a payload. Documentation for WiiMote Report IDs has not been published, but some Report IDs have been identified and documented in the WiiBrew Wiki (WiiBrew, 2011).

As with all networked devices, the WiiMote Bluetooth interface is identified by a unique **MAC** (**media access control**) address. A MAC address is a sequence of 6 bytes, usually written something like “00:1E:35:3B:7E:6D” using hexadecimal notation. The MAC ad-

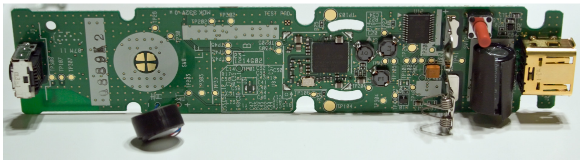


Figure B.12: Nintendo Wii Remote exposed (WiiBrew, 2011).

dress is not visible on the WiiMote device itself, but can be found by pairing to a desktop computer.

Bluetooth pairing becomes difficult when multiple devices are present, as may be the case in a laboratory. This issue is resolved by pairing according to the MAC address of the specific device one wishes to pair. Unfortunately, the MAC address is not visible on the WiiMote itself; it can be found by manually pairing the WiiMote using the standard Windows Bluetooth device paring. Once paired, the device properties will show the MAC address. The MAC address can then be printed on the side of the device for future reference.

B.4.2 Exercises

1. This exercise requires that you read about the Bluetooth interface to the WiiMote. Part of the goal of this exercise is to get familiar with how to find the required information. For each question, first visit the unofficial WiiBrew Wiki to locate the relevant command sequence, then reference the official Bluetooth HID documentation to decode its meaning. Use hexidecimal (0x__) notation for all numeric values.

   (a) What is the command to enable LEDs 1 and 3? Identify the Transaction Header (Transaction Type and Parameter), Report ID, and Payload (if present).

   (b) Given the sequence to enable LEDs 1 and 3, how can it be modified to also enable the rumble motor?

   (c) What is the command to enable continuous reporting of the core buttons and the three axes of the accelerometer (even if the device is at rest)? Identify the Transaction Header (Transaction Type and Parameter), Report ID, and Payload (if present).

(d) What is the Transaction Header transmitted by the WiiMote when sending a report of its sensors?

B.5 Xilinx MicroBlaze

Xilinx MicroBlaze is a soft-core processor implemented on an FPGA. We provide the MicroBlaze processor as an FPGA fixed personality for sbRIO,

```
cpsLab/drivers/FPGA bitfiles/microBlazeFixedPersonality.lvbitx
```

Useful references:

1. MicroBlaze product page (Xilinx, 2011)
2. MicroBlaze Processor Reference Guide (Xilinx, 2010d)
3. MicroBlaze Interrupts and the EDK (Hickok, 2009)

Related reading:

1. Introduction to Embedded Systems (Lee and Seshia, 2011)
   (a) Chapter 7: Embedded Processors, §7.1.1 (Microcontrollers)
   (b) Chapter 7: Embedded Processors, §8.2.1 (Memory Maps)
   (c) Chapter 7: Embedded Processors, §8.2.2 (Register Files)

B.5.1 Debugging MicroBlaze on sbRIO

The National Instruments sbRIO does not have an easily accessed JTAG interface, but JTAG support can easily be added by implementing the protocol on the FPGA. With this, the Xilinx SDK can download and debug applications for MicroBlaze running on the sbRIO FPGA. The JTAG interface is part of the fixed personality.

When using the Analog RMC, the correct wiring to a JTAG device is shown in Table B.1. These connections are used by the JTAG device to program and debug MicroBlaze.

To download the FPGA fixed personality for the MicroBlaze core, open RIO Device Setup from the Windows start menu under “Programs, National Instruments, NI-RIO, RIO Device Setup”, enter the IP address of your sbRIO, and download the MicroBlaze fixed personality

JTAG Signal	**JTAG Color**	**sbRIO Pin**
VREF	red	5V
GND	black	GND
MISO	purple	DIO 43
MOSI	white	DIO 42
TCK	yellow	DIO 41
TMS	green	DIO 40

Table B.1: JTAG to sbRIO wiring diagram.

```
cpsLab/drivers/FPGA bitfiles/microBlazeFixedPersonality.lvbitx
```

Once the bitfile has been successfully deployed, switch to the Device Settings tab, check the box “Autoload VI on device reboot”, press “Apply Settings”, and then manually reset sbRIO. MicroBlaze should now be deployed and running on the FPGA.

C

Lab Setup

Contents

This appendix describes how to configure hardware and software to support all of the laboratory exercises in this text. The exercises are designed to be modular, and where possible, we indicate steps that may be omitted if certain modules are to be excluded. We emphasize this is only a guideline, and that a proper laboratory configuration is a cooperative effort between instructors and laboratory administrators.

C.1 Software Tools

C.1.1 Software Recommended for a Lab Workstation

1. National Instruments LabVIEW 2011
   (a) LabVIEW Real-Time Module
   (b) LabVIEW Robotics Module
   (c) LabVIEW Statecharts Module
   (d) NI Device Drivers - NI-RIO 4.0

C.1.2 Software Tools Installation

LabVIEW

1. Install LabVIEW 2011 from the source DVD, making sure to select the recommended modules.

2. Run the NI Update Service and apply all available patches.

C.2 Software Infrastructure

A series of software libraries have been developed and are provided as a starting point for lab curriculum.

C.2.1 Single-Board RIO Software Architecture

As discussed in Chapter B: Lab Hardware, §B.3 (National Instruments Single-Board RIO 9606), the sbRIO is a heterogenous, reconfigurable architecture. For many labs, the sbRIO is configured to perform as a traditional microcontroller with fixed hardware peripherals and a programmatic interface. The hardware peripherals are implemented as a fixed-personality on the FPGA, and user programs interact with these peripherals through a programming interface shown in Fig. C.1:

The **Berkeley FPGA fixed-personality** used in this lab is named for its development at the University of California, Berkeley. The Berkeley FPGA personality is written in LabVIEW FPGA, and the source code is downloadable from the Jensen, Lee, and Seshia website. The FPGA source is compiled into a **LabVIEW FPGA bitfile**, a binary file that may be used to reconfigure the FPGA. It is generally easier to redistribute the bitfile, since FPGA compile times can be lengthy, and the binary is completely self-contained. Complex peripherals such as UART and I^2C communication ports have an additional software interface implemented in both LabVIEW G code and C code that accompanies the Berkeley FPGA fixed-personality.

Additional software libraries include an interface to the iRobot Create, which builds on the Real-Time API. To run a user library, you must have either the source code or compiled bitfile for the Berkeley FPGA fixed-personality, the Real-Time API, and the user library itself.

Compiling and Running the Berkeley FPGA Fixed-Personality

This is an optional step; it is generally easier to use the compiled bitfile that is distributed with each lab module.

1. The Berkeley FPGA fixed-personality source is included in the lab resource files, in the folder

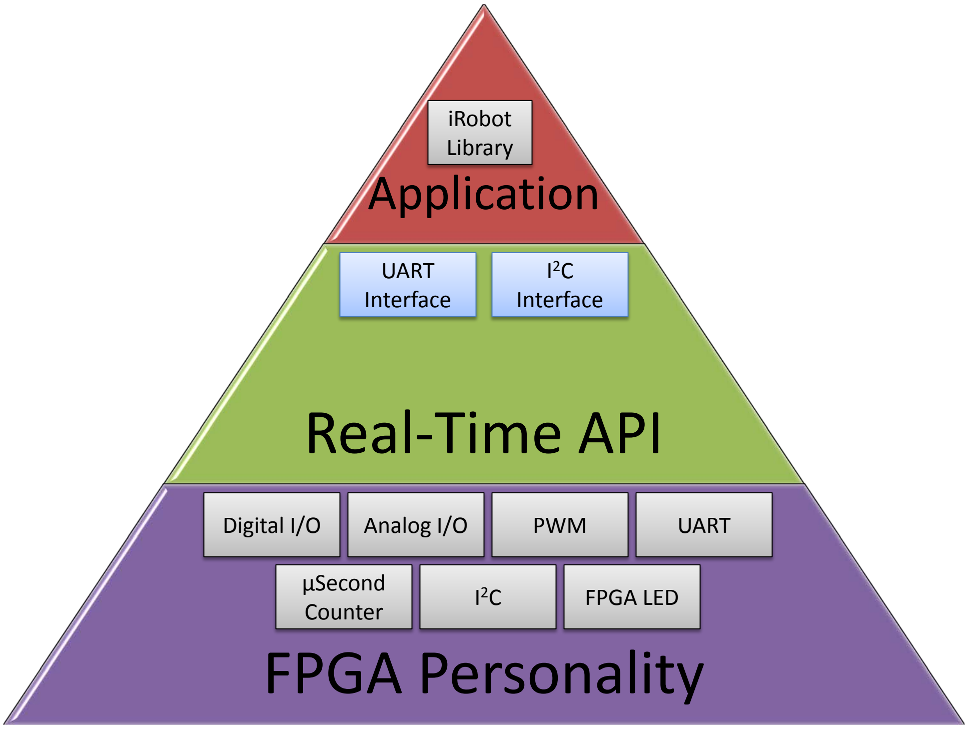


Figure C.1: Software architecture for the sbRIO.

```
cpsLab/drivers/LabVIEW/sbRIO
```

.

2. Open the project file `berkeleyFixedPersonality_9606.lvproj`.

3. Right-click on the sbRIO target and change the IP address to the address of your controller.

4. Open **Berkeley Fixed Personality 9606.vi.vi**. This is the top-level VI for the FPGA.

5. Run the VI. This will begin the FPGA synthesis and compilation process, which can take an hour or more on some machines. When the compilation is complete, the VI will be programmed to the FPGA and the VI will enter interactive mode.

C.3 Lab Workstation Hardware

C.3.1 Recommended Hardware

1. National Instruments Single-Board RIO (National Instruments, 2011d)
   (a) sbRIO-9606 embedded microcontroller
   (b) PS-2, PS-3, or PS-15 power supply
   (c) Custom analog module (C.3.2)
2. Asus WL-330ge (ASUS, 2011) wireless router (optional for wireless programming and debugging of sbRIO)
3. Custom power adapter (optional for switching power to sbRIO and a wireless router) (C.3.4)
4. iRobot Create (iRobot, 2011)
5. Nintendo Wii Remote (Nintendo, 2011)
6. Analog Devices ADXL-335 (Analog Devices, 2011) analog accelerometer

C.3.2 Connect the Analog RMC

The analog RMC is documented in Appendix B: Lab Hardware, §B.3.2 (Analog RMC). Refer to the NI sbRIO-9605/9606 OEM Operating Instructions and Specifications, §"Mounting the NI sbRIO-960x" for instructions on how to mate the analog RMC (Fig. C.2-C.3).

C.3.3 Connect a Workstation to sbRIO

A 5-minute video tutorial on connecting sbRIO is available from `http://zone.ni.com/wv/app/doc/p/id/wv-745`.

1. Power sbRIO according to NI sbRIO-9605/9606 OEM Operating Instructions and Specifications, §"Powering the NI sbRIO Device".

2. For wired communication, connect sbRIO to your network according to NI sbRIO-9605/9606 OEM Operating Instructions and Specifications, §"Connecting the NI sbRIO Device to a Network".

3. For wireless communication, connect sbRIO directly to your wireless router. The router will be used in a bridge configuration to connect the wired ethernet port on sbRIO to a wireless network. Follow the instructions in the ASUS WL-330gE User's Manual

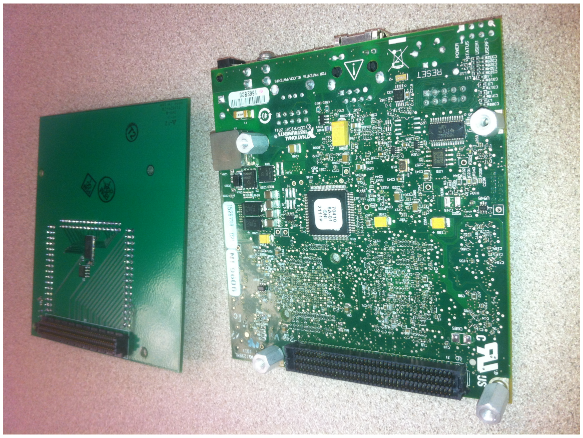


Figure C.2: Spacers placed in preparation for the analog RMC.

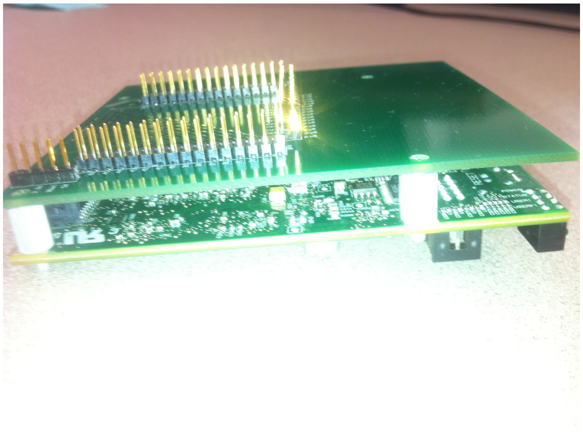


Figure C.3: Properly mounted and mated analog RMC.

§4.2.3 “Ethernet Adapter Mode” to configure the router for “Ethernet Adapter Mode”, which will create a transparent link from the sbRIO to your wireless network.

An alternate configuration is to use the wireless router as an access point; this is slightly more complicated, and we refer the user to the ASUS WL-330gE User’s Manual for instructions.

4. Discover sbRIO using the NI Measurement and Automation Explorer (MAX). You may find the IP address of the sbRIO using this tool, and also set it should you want a static IP address. The typical application is to use DHCP, which works in most networks and also when the controller is directly connected the host without a router or switch.

If the sbRIO does not appear on your network, visit the troubleshooting page Troubleshoot When NI CompactRIO Does Not Appear in Measurement & Automation Explorer (MAX).

5. To verify your computer is able to communicate with sbRIO, use the system `ping` tool to ping the IP address you discovered (or set) in MAX.

C.3.4 Build the Cal Climber

The Cal Climber (Jensen et al., 2011) is a cyber-physical system developed at the University of California, Berkeley, for use in embedded and robotics courses. Based on the iRobot Create consumer robotics platform, it includes an embedded controller, analog accelerometer, and optionally a wireless adapter. A custom power adapter can be used to power the sbRIO and wireless adapter. The power adapter PCB layout, build of materials, and publication exports are downloadable from

`http://LeeSeshia.org/lab/releases/0.05/setup/irobotDCadapter.zip`

A complete wiring diagram is shown in Fig. C.10 (custom power adapter used) and Fig. C.11 (no custom power adapter)

1. If using the custom power adapter to switch power from the sbRIO and/or wireless router (Fig C.5):

   (a) Mount the adapter in the cargo bay of the iRobot Create.
   (b) Connect iRobot Create cargo bay pin 12 (+15V switched) to J1 pin 1 (+15V in) on the power adapter.

   (c) Connect iRobot Create cargo bay pin 25 (GND) to J1 pin 2 (GND) on the power adapter.

2. Mount the sbRIO in the cargo bay of the iRobot Create.

3. Power the sbRIO (Fig. C.6):

   (a) If using the custom power adapter (Fig. C.6(a)):
      i. Connect power adapter terminal J3 pin 1 (+15V out) to the positive terminal of the sbRIO power connector.
      ii. Connect power adapter terminal J3 pin 2 (GND) to the C (common) terminal of the sbRIO power connector.

   (b) If instead powering directly from the iRobot Create (Fig. C.6(b)):
      i. Connect iRobot Create cargo bay pin 12 (+15V switched) to the positive terminal of the sbRIO power connector.
      ii. Connect iRobot Create cargo bay pin 25 (GND) to the C (common) terminal of the sbRIO power connector.
      iii. The sbRIO should now be powered when the iRobot Create is turned on.

4. If using wireless communication, add the wireless router (Fig. C.7).

   (a) Mount the wireless router in the cargo bay of the iRobot Create.
   (b) If using the custom power adapter: use a spliced wire with red (positive) and black (ground) leads (Fig. C.7(a)).
      i. Connect power adapter terminal J2 pin 1 (+5V out) to the positive lead of the router.
      ii. Connect power adapter terminal J2 pin 2 (GND) to the ground lead of the router.
   (c) If instead powering directly from the sbRIO, use a USB to +5V tunnel adapter (Fig. C.7(b)).

5. Mount the accelerometer in the cargo bay of the iRobot Create. A useful orientation is such that uphill orientation yields equal acceleration measurements on the x and y axes (Fig. C.4).

6. The accelerometer can be powered from the sbRIO, and its output should be wired to the sbRIO Analog RMC (Fig. C.8).

   (a) Wire the accelerometer Vs (source power) pin to the +3.3V pin on the sbRIO Analog RMC.
   (b) Wire the accelerometer COM (GND) pin to a ground (GND) pin on the analog port of the sbRIO Analog RMC.

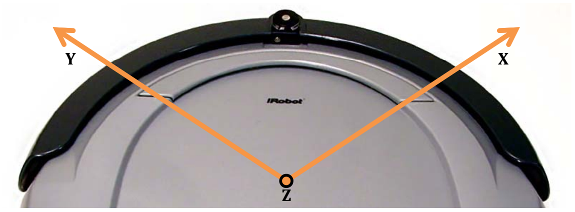


Figure C.4: Suggested mounting of the accelerometer on the Cal Climber. (Moore, 2006)

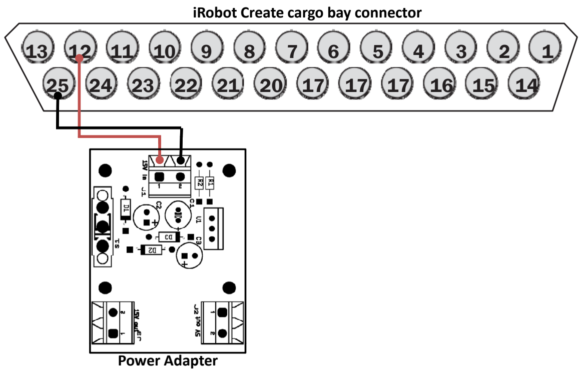


Figure C.5: Connecting the iRobot Create to the custom power adapter. Pinout diagrams replicated from the iRobot Create Open Interface (OI) Specification (iRobot, 2006a).

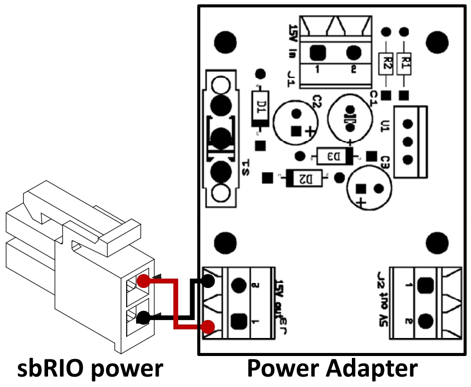
(c) Wire the accelerometer Xout (X axis output) pin to ch0 (analog input channel 0) on the sbRIO Analog RMC.

(d) Wire the accelerometer Yout (Y axis output) pin to ch1 (analog input channel 1) on the sbRIO Analog RMC.

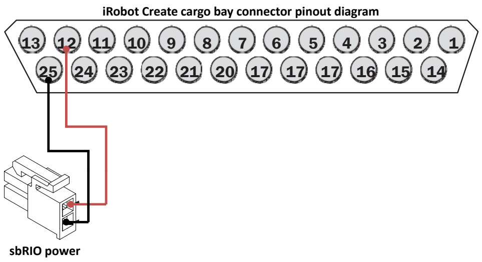
7. Connect the UART serial port (Fig. C.9).

(a) Connect iRobot Create cargo bay pin 1 (UART Rx) to sbRIO Analog RMC pin 36 (UART0 Tx).

(b) Connect iRobot Create cargo bay pin 2 (UART Tx) to sbRIO Analog RMC pin 37 (UART0 Rx).

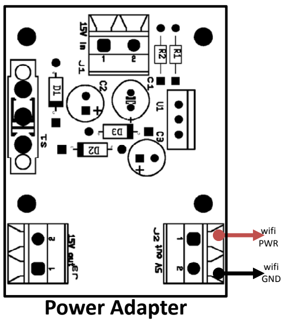


(a) sbRIO powered by the custom power adapter.

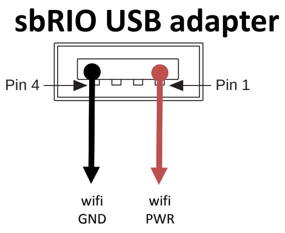


(b) sbRIO powered directly from the iRobot Create.

Figure C.6: Powering sbRIO from the iRobot Create battery. Pinout diagrams replicated from the iRobot Create Open Interface (OI) Specification (iRobot, 2006a).



(a) Router powered by the custom power adapter.



(b) Router powered by the sbRIO USB port.

Figure C.7: Powering the wireless router.

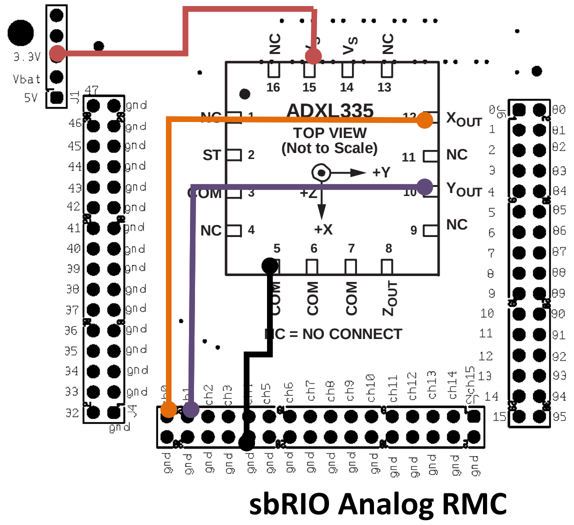


Figure C.8: Wiring ADXL-335 accelerometer to sbRIO Analog RMC. Pinout diagrams replicated from the Analog Devices ADXL-335 Datasheet (Analog Devices, 2010).

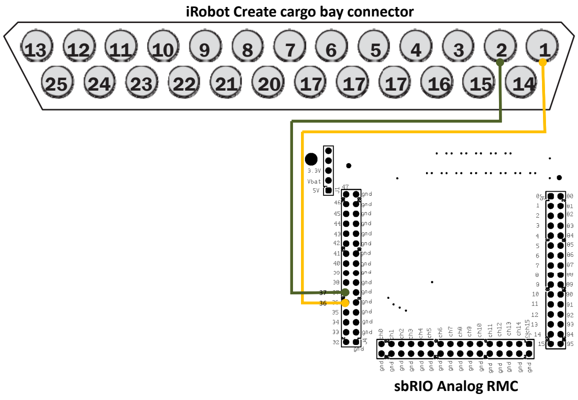


Figure C.9: Wiring iRobot UART to the sbRIO Analog RMC. Pinout diagrams replicated from the iRobot Create Open Interface (OI) Specification (iRobot, 2006a).

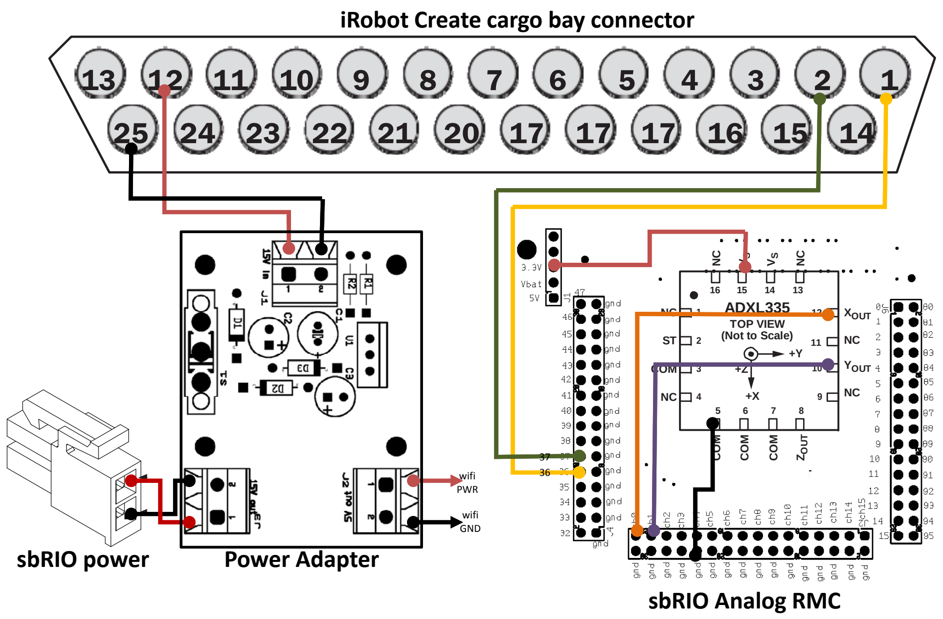


Figure C.10: Cal Climber wiring diagram. A custom power adapter switches power to the sbRIO and the wireless router. Pinout diagrams replicated from the iRobot Create Open Interface (OI) Specification (iRobot, 2006a) and the Analog Devices ADXL-335 Datasheet (Analog Devices, 2010).

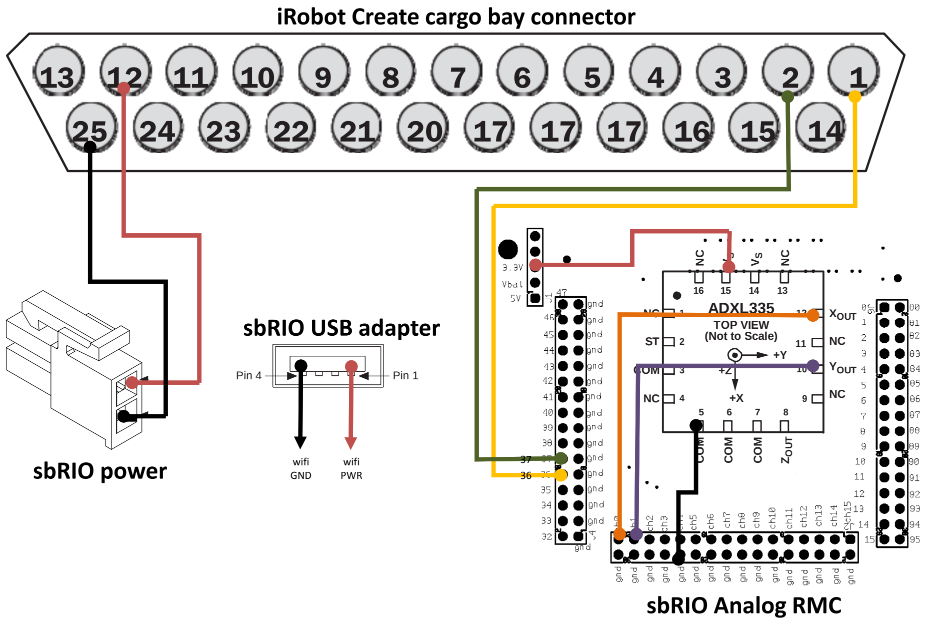


Figure C.11: Cal Climber wiring diagram. All devices are powered directly from the iRobot or sbRIO. Pinout diagrams replicated from the iRobot Create Open Interface (OI) Specification (iRobot, 2006a) and the Analog Devices ADXL-335 Datasheet (Analog Devices, 2010).

Bibliography

Analog Devices, 2010: ADXL335 Datasheet. Rev. B, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/AnalogDevices_10_Adxl335Datasheet.pdf. Available from: http://www.analog.com/static/imported-files/data_sheets/ADXL335.pdf.

—, 2011: ADXL335 product page. Available from: http://www.analog.com/en/mems-sensors/inertial-sensors/adxl335/products/product.html.

ASUS, 2011: ASUS WL-330gE product page. Available from: http://www.asus.com/Networks/Wireless_Routers/WL330gE/.

Bluetooth Special Interest Group, 2003: Bluetooth HID Profile. Version 1.0 Adopted. Available from: https://www.bluetooth.org/Technical/Specifications/adopted.htm.

Brooks, C., 2008: Project management instructions. Mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/Brooks_08_ProjectManagement.doc. Available from: http://technology.berkeley.edu/cio/tpo/project/pmresources/tools/Project_Charter_Instructions.doc.

—, 2012: Eecs instructional public documentation - svn help. Available from: https://inst.eecs.berkeley.edu/cgi-bin/pub.cgi?file=svn.help.

Dudek, G. and M. Jenkin, 2000: *Computational Principles of Mobile Robotics*. Cambridge University Press.

Fisher, C. J., 2010: Using an Accelerometer for Inclination Sensing. Analog Devices application note AN-1057 rev. 0, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/Fisher_10_UsingAnAccelerometerForInclinationSensing.pdf. Available from: http://www.analog.com/static/imported-files/application_notes/AN-1057.pdf.

Harel, D., 1987: Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, **8(3)**, 231–274.

Hickok, T., 2009: MicroBlaze Interrupts and the EDK. Mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/Hickok_09_MicroBlazeInterruptsEDK.pdf. Available from: http://courseware.ee.calpoly.edu/cpe-329/EDK_Resources/th_MicroBlaze_interrupts.pdf.

iRobot, 2006a: iRobot Create Open Interface (OI) Specification. version 2, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/iRobot_06_CreateOpenInterfaceSpecification.pdf. Available from: http://www.irobot.com/filelibrary/create/Create%20Open%20Interface_v2.pdf.

—, 2006b: iRobot Create Owner's Guide. version 430.06, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/iRobot_06_CreateOwnersGuide.pdf. Available from: http://www.irobot.com/filelibrary/create/Create%20Manual_Final.pdf.

—, 2011: iRobot Create product page. Available from: http://store.irobot.com/create.

Jensen, J. C., E. A. Lee, and S. A. Seshia, 2011: An introductory capstone design course on embedded systems. In *IEEE International Symposium on Circuits & Systems (ISCAS), special session: Design, Project, & Learning Technology Innovations in Circuits, Signals, & Systems Education*, Rio de Janeiro, Brazil, pp. 1199–1202. doi:10.1109/ISCAS.2011.5937784.

Kodosky, J., J. MacCrisken, and G. Rymar, 1991: Visual programming using structured data flow. In *IEEE Workshop on Visual Languages*, IEEE Computer Society Press, Kobe, Japan, pp. 34–39.

Kung, S., L. Onken, and S. Large, 2011: Tortoisesvn. Available from: `http://tortoisesvn.net/docs/release/TortoiseSVN_en/`.

Lee, E. A., 2006: The problem with threads. Tech. Rep. UCB/EECS-2006-1, EECS Department, University of California, Berkeley. Available from: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.html`.

Lee, E. A. and S. A. Seshia, 2010: An introductory textbook on cyber-physical systems. In *Proc. Workshop on Embedded Systems Education (WESE)*, pp. 1–6. `doi:10.1145/1930277.1930278`.

—, 2011: *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. LeeSeshia.org, Berkeley, CA. Available from: `http://LeeSeshia.org`.

Lee, E. A., S. A. Seshia, and J. C. Jensen, 2012: EECS 149 - Introduction to Embedded Systems. course website. Available from: `http://chess.eecs.berkeley.edu/eecs149`.

Moore, L., 2006: First generation roomba. Creative Commons BY-SA 3.0, via Wikipedia. Available from: `http://en.wikipedia.org/wiki/File:Roomba_original.jpg`.

National Instruments, 2009: MathScript RT Module Functions. P/N #371361F-01. Available from: `http://zone.ni.com/reference/en-XX/help/371361F-01/lvtextmath/msfunc_classes/`.

—, 2010a: Developing Algorithms Using LabVIEW MathScript RT Module: Part 1 - The LabVIEW MathScript Node. Mirrored from: `http://LeeSeshia.org/lab/releases/0.05/documents/NationalInstruments_10_DevelopingAlgorithmsUsingLabVIEWMathScriptRtModule1.pdf`. Available from: `http://zone.ni.com/devzone/cda/tut/p/id/3256`.

—, 2010b: Developing Applications with the NI LabVIEW Statechart Module. Mirrored from: `http://LeeSeshia.org/lab/releases/0.05/documents/NationalInstruments_10_DevelopingApplicationsWithNILabVIEWStatechartModule.pdf`. Available from: `http://zone.ni.com/devzone/cda/tut/p/id/6194`.

—, 2010c: Getting Started with LabVIEW. P/N #373427G-01, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/NationalInstruments_10_GettingStartedWithLabVIEW.pdf. Available from: http://www.ni.com/pdf/manuals/373427g.pdf.

—, 2010d: LabVIEW Quick Reference Card. P/N #373353D-01, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/NationalInstruments_10_LabVIEWQuickReferenceCard.pdf. Available from: http://www.ni.com/pdf/manuals/373353d.pdf.

—, 2011a: LabVIEW 2011 Help. P/N #371361H-01. Available from: http://zone.ni.com/reference/en-XX/help/371361H-01/.

—, 2011b: LabVIEW Control, Design, and Simulation Module Help. P/N #371894F-01. Available from: http://zone.ni.com/reference/en-XX/help/371894F-01/.

—, 2011c: LabVIEW Statechart Module Help. P/N #372103D-01. Available from: http://www.ni.com/pdf/manuals/372103d.zip.

—, 2011d: NI sbRIO-9605/9606 OEM Operating Instructions and Specifications. P/N #373378A-01, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/NationalInstruments_11_sbRIO9606.pdf. Available from: http://www.ni.com/pdf/manuals/373378a.pdf.

—, 2011e: NI sbRIO-9606 product page. Available from: http://sine.ni.com/nips/cds/view/p/lang/en/nid/210003.

—, 2011f: NI Single-Board RIO Setup and Services. Available from: http://www.ni.com/singleboard/setup/.

—, 2011g: What is the MathScript RT Module? tutorial, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/NationalInstruments_11_WhatIsTheMathScriptRtModule.pdf. Available from: http://zone.ni.com/devzone/cda/tut/p/id/6206.

—, 2012a: LabVIEW product page. Available from: http://ni.com/LabVIEW.

—, 2012b: MathScript product page. Available from: http://www.ni.com/labview/mathscript/.

Nintendo, 2011: Controllers at Nintendo. Available from: http://www.nintendo.com/wii/console/controllers.

Orlando, G., 2010: Wii remote image. Creative Commons BY-SA 3.0, via Wikipedia. Available from: http://en.wikipedia.org/wiki/File:Wiimote.png.

Ozyagcilar, T., 2011: Implementing a tilt-compensated ecompass using accelerometer and magnetometer sensors. Analog Devices application note AN-4248 rev. 2, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/Ozyagcilar_11_ImplementingCompass.pdf. Available from: http://www.freescale.com/files/sensors/doc/app_note/AN4248.pdf.

SparkFun, 2009: Adxl335 schematic. rev. 13, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/SparkFun_09_Adxl335Schematic.pdf. Available from: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/BreakoutBoards/ADXL335_v13.pdf.

Stoops, M., 2008: Wii remote uncovered. Copyright © Michael Stoops, GNU Free Documentation License 1.2. Available from: http://wiibrew.org/wiki/File:Wii_Remote_uncovered.jpg.

Texas Instruments, 2010: 12/10/8-Bit, 1 MSPS, 16/12/8/4-Channel, Single-Ended, MicroPower, Serial Interface ADCs. SLAS605A, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/TexasInstruments_10_ADS7953.pdf. Available from: http://www.ti.com/lit/ds/slas605a/slas605a.pdf.

Tortoise SVN Team, 2012: Tortoise SVN. product page. Available from: http://tortoisesvn.net/.

Washington, C., 2009: Graphical System Design with LabVIEW Statecharts. National Instruments webcast. Available from: http://zone.ni.com/wv/app/doc/p/id/wv-213.

WiiBrew, 2011: Wiimote. November 2011 mirror available from: http://LeeSeshia.org/lab/releases/0.05/documents/WiiBrew_11_WiiMote_112811.pdf. Available from: http://wiibrew.org/wiki/Wiimote.

Wikipedia, 2012a: Milestone (project management). Available from: http://en.wikipedia.org/wiki/Milestone_(project_management).

—, 2012b: Project charter. Available from: http://en.wikipedia.org/wiki/Project_charter.

—, 2012c: Wii remote. Mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/Wikipedia_12_WiiMote_020112.pdf. Available from: http://en.wikipedia.org/wiki/Wii_Remote.

—, 2012d: Work breakdown structure. Available from: http://en.wikipedia.org/wiki/Work_breakdown_structure.

Wind River Systems, 2006: Workbench User's Guide. 2.5 (VxWorks version P/N # DOC-15745-ZD-00). Available from: http://LeeSeshia.org/lab/releases/0.05/documents/WindRiver_06_Workbench2.5UserGuide.pdf.

—, 2012: Workbench product page. Available from: http://www.windriver.com/products/workbench/.

Xilinx, 2010a: Embedded System Tools Reference Manual. Version UG111 12.4, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/Xilinx_10_EmbeddedSystemToolsReferenceManual.pdf. Available from: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/est_rm.pdf.

—, 2010b: LogiCORE IP XPS Interrupt Controller. Version 2.01a, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/Xilinx_10_LogiCoreIpXpsInterruptController.pdf. Available from: http://www.xilinx.com/support/documentation/ip_documentation/xps_intc.pdf.

—, 2010c: LogiCORE IP XPS Timer/Counter. Version 1.02a, mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/Xilinx_10_LogiCoreIpXpsTimerCounter.pdf. Available from: http://www.xilinx.com/support/documentation/ip_documentation/xps_timer.pdf.

—, 2010d: MicroBlaze Processor Reference Guide. version UG081 (v11.4), mirrored from: http://LeeSeshia.org/lab/releases/0.05/documents/Xilinx_10_MicroBlazeProcessorReferenceGuide.pdf. Available from: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/mb_ref_guide.pdf.

—, 2011: MicroBlaze Soft Processor Core. Available from: `http://www.xilinx.com/tools/microblaze.htm`.

Index